A posteriori log analysis and security rules violation detection

Farah Dernaïka

DOCTORAT BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

THESE DE DOCTORAT DE

L'ÉCOLE NATIONALE SUPERIEURE MINES-TELECOM ATLANTIQUE
BRETAGNE PAYS DE LA LOIRE - IMT ATLANTIQUE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par

Farah DERNAIKA

A posteriori log analysis and security rules violation detection

Thèse présentée et soutenue à Cesson-Sévigné, le 13 Octobre 2020
Unité de recherche : Lab-STICC
Thèse N° : 2020IMTA0210

Rapporteurs avant soutenance :

Alban Gabillon	Professeur, Université de la Polynésie Française
Joaquin Garcia-Alfaro	Professeur, Télécom SudParis

Composition du Jury :

Président :	Mohand-Said Hacid	Professeur, Université Claude Bernard Lyon
Examinateurs :	Alban Gabillon	Professeur, Université de la Polynésie Française
	Joaquin Garcia-Alfaro	Professeur, Télécom SudParis
	Romain Laborde	Maître de conférence (HDR), Université Paul Sabatier
	Olivier Raynaud	Consultant, Almerys
	Nora Cuppens-Boulahia	Professeur, Polytechnique Montréal
	Frédéric Cuppens	Professeur, Polytechnique Montréal
Dir. de thèse :	Eric Totel	Professeur, IMT Atlantique

To my parents Mounir and Jinane Dernaika

Abstract

Traditional access control models prevent violations of the security policy by blocking any unauthorized action. However, in sensitive environments, such as the healthcare domain, a lot of unanticipated situations may occur, imposing the need to have an immediate access to information resources without risk of rejection. Therefore, the deployment of a more flexible access control model is needed.

The a posteriori access control mode consists in monitoring users' actions in order to detect potential violations of the security policy and apply sanctions and/or reparations. This monitoring process is usually based on log analysis, where all access evidences persist. It must also be combined with a deterrent sanction policy so that users are not tempted to violate the security policy. In the literature, this kind of security check was divided into three stages that are: log processing, log analysis, and accountability. The first step is meant to extract relevant information from logs, that are analyzed later on in the second phase to detect violations. When abuse of privilege is assumed, the process ends by assigning responsibilities by applying sanctions and remedies if necessary.

In this thesis, we cover these three areas of the a posteriori access control by providing novel solutions, and we introduce some new aspects that were not addressed previously. We propose new means to extract relevant information from logs by using a semantic mediator and treat the semantic enrichment of logs. Moreover, we leverage the a posteriori access control to include temporal compliance, and we consider the violations that can be caused by both regular users and administrators. Finally, we propose an accountability mechanism for the a posteriori access control.

Résumé

Les modèles de contrôle d'accès traditionnels empêchent les violations de la politique de sécurité en bloquant toute action non autorisée. Cependant, dans des environnements sensibles, comme dans le domaine de la santé, de nombreuses situations imprévues peuvent se produire, imposant la nécessité d'avoir un accès immédiat aux ressources d'information sans risque de rejet. Il est donc nécessaire de déployer un modèle de contrôle d'accès plus flexible.

Le mode de contrôle d'accès a posteriori consiste à surveiller les actions des utilisateurs afin de détecter d'éventuelles violations de la politique de sécurité et d'appliquer des sanctions et/ou des réparations. Ce processus de surveillance est généralement basé sur l'analyse des fichiers journaux, où toutes les preuves d'accès persistent. Il doit également être associé à une politique de sanctions dissuasive afin que les utilisateurs ne soient pas tentés de violer la politique de sécurité. Dans la littérature, ce type de contrôle de sécurité a été divisé en trois étapes qui sont : le traitement des logs, l'analyse des logs, et l'imputabilité. La première étape vise à extraire des informations pertinentes des logs, qui sont analysées plus tard dans la deuxième phase pour détecter les violations. Lorsqu'un abus de privilège est présumé, le processus finit par l'attribution des responsabilités en appliquant des sanctions et des réparations si nécessaire.

Dans cette thèse, nous couvrons ces trois domaines du contrôle d'accès a posteriori en apportant de nouvelles solutions, et nous introduisons des nouveaux aspects qui n'avaient pas été abordés auparavant. Nous proposons une nouvelle méthode pour extraire les informations pertinentes des logs en utilisant un médiateur sémantique et nous traitons l'enrichissement sémantique des logs. En outre, nous étendons le contrôle d'accès a posteriori pour inclure la conformité temporelle, et nous prenons en

considération les violations qui peuvent être causées par les utilisateurs réguliers ainsi par les administrateurs. Enfin, nous proposons un mécanisme d'imputabilité pour le contrôle d'accès a posteriori.

Acknowledgments

First, I would like to thank my PhD directors, Nora and Frédéric Cuppens for giving me the opportunity of doing a PhD with such great people like them. They have been more than just supervisors, they were like my parents, always supporting and encouraging me. Moreover, I surely thank them for sharing their great knowledge and research experience to always produce the best.

I would also like to thank my supervisor Olivier Raynaud for his guidance during my thesis. I am so lucky to work with him as he does not only care about the quality of the work, but also about the well-being of his students. Another special thanks to professor Eric Totel for all his efforts to organize my thesis defense. I would also like to thank all the jury members for being a part of my thesis especially the professors Alban Gabillon and Joaquin Garcia-Alfaro for reporting my thesis and giving their recommendations.

Next, I would like to thank all my colleagues and friends that became a part of my family. I will never forget these most beautiful 3 years I spent with them. A special thanks to my girl squad: Routa, Rahaf, Tania, and Nisrine.

Most importantly, I would like to thank my family, especially my parents Mounir and Jinane Dernaika, to whom I dedicate this thesis. They are the reason for who I am today, and I will always be grateful to them. I would also like to thank my brother Abdel Aziz, my sister Ouhayla, and my brother in law Sami Ayoubi, for their love and support.

Last but not least, I express all my gratitude to my life partner Taha Merhebi for his unconditional support and endless patience. He has been always by my side, and I am so excited for our upcoming life together.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

For several years now, information and communication technologies have been revolutionizing all sectors, whether they are industrial, commercial, administrative, or medical. As these businesses become digital through this data-driven transformation, the value of data increases to be one of the most important assets that an association can have. Therefore, to ensure the security of data, access control came along to protect the information system against any activity that could lead to a security breach or any accidental and malicious threats by regulating access to data. That being said, access control is an essential security requirement that needs to be ensured in organizations to assure the confidentiality, integrity, and availability in their information systems.

Access control requires the enforcement of access control policies that consist in a set of rules that define access control requirements (permissions, prohibitions) relating to the actions performed by a user in an information system. Although the criticality of access control is recognized by organizations, its enforcement may differ between one and another. In fact, the way of maintaining access control requirements depends on the environment in which it is being implemented.

Traditional access control models verify users' privileges before granting them access to information resources to avoid misuse of privileges. Through authentication and authorization, access control policies check if the users are who they claim they are and have appropriate access to the concerned resource. This a priori enforcement

of security policies is preventive since it restricts access to information resources if the conditions defined in the policy are not fulfilled and thus, can be inadequate in dynamic environments where access decisions may depend on contextual conditions. For instance, in the healthcare domain, a lot of emergencies may occur imposing the need of having an immediate access; hence, to avoid having serious consequences, security controls in the corresponding information systems must not block certain decisions and actions of users. In consequence, it is a prerequisite to take into account the organization's uses and practices, so that the deployed security solution is not perceived as a constraint for users with a significant risk of rejection that can impose undesirably high computational costs. In this regard, the use of a more flexible access control appeared to be convenient, where it is possible to identify and trace these decisions and actions, in order to detect possible breaches of the security policy put in place and set responsibilities.

The problematic of the a posteriori access control is fairly recent, and its prime concerns are auditability and accountability in order to detect potential violations of the security policy to prevent future misuse of privileges. In this type of access control, access to information is given based on a trust level offered to the user, where this latter is deterred from committing policy violations by a mechanism that assures the traceability of his actions, and that applies sanctions in case of an abuse of privileges. Generally, this monitoring process starts by analyzing logs since they trace and record all the executed actions in the information system. Recognizing the importance of logs, the National Institute of Standards and Technology, USA issued best practices and recommendations for computer security log management [115]. Thus, logs are among the first data sources that information security specialists consult for forensics when they suspect that something went wrong. In contrast, the primary reference of this a posteriori analysis is the security policy, as the goal is to detect potential violations of this policy. Therefore, the a posteriori access control works on the reconciliation between policy rules and logged actions in order to verify whether access rules are being fully respected or not. As a result, previous a posteriori access control approaches defined the compliance verification process as composed of three main components that are logging, log analysis, and accountability [38, 48, 10]. While logging provides evidence of users' actions, log analysis identifies abnormal ones, and

then the accountability component applies penalties to the user in case no legitimate reason for his misbehavior was found. These three main steps need to be carefully addressed since any error in the process can lead to complex problems with legal, ethical, and social dimensions.

In this thesis, we study an approach based on an a posteriori access control to detect any violation of the security policy. We thus present the difficulties that reside in this kind of security check, as well as our proposed solutions in the following sections.

1.2 Problem Statement

The first step of the a posteriori access control consists in extracting relevant information from log files in order to be analyzed and then detect violations. Nevertheless, the first difficult challenge we face when treating and analyzing logs, is the multiple log file formats. This is generally due to the different types of log sources such as Application server, Web server, Database, etc. To address this issue, several log normalization methods have been proposed in the literature that intended to process logs to put them in a unique format. Unfortunately, none of these efforts were able to attract sufficient attention, namely for their shortcomings regarding their runtime and memory consumption that make them unsuitable for environments with a large log volume. Therefore, we need to deal with those format differences by providing a module that extracts information in terms of the security policy from the different source logs without imposing a particular format.

Another important issue that makes the a posteriori investigations doubtful is that log analysis is based essentially on the expertise of the person who performs it. For example, the system administrator can use a generic list of security checks that is not necessarily adapted to the target system. Thus, several efforts have been made to detect anomalies from logs such as process mining [203], and machine learning techniques [60] that were also integrated into some log analysis tools, e.g., Splunk [132]. Yet, these methods always require human intervention for further analysis to decide what is really normal vs. abnormal. When it comes to access control, the security policy is the judge. Consequently, it is necessary to define a reference

format for the security policy in order to facilitate the detection of potential violations. Once the security policy is defined, we need to establish links between the extracted logged events and the rules defined in the policy to be able to proceed with the analysis and detect violations. However, the entities relating to the subject and object involved in the logged access may differ from the ones that are defined in the security policy, particularly, in case of an expressive policy [51]. Therefore, controlling the respect of the policy a posteriori must be based on effective monitoring mechanisms to make decisions about policy violations. These mechanisms must rely on means that complement logs with additional information to allow the comparison between the actions performed in the logs and the permissions defined in the policy, as well as assessing their compliance.

Furthermore, in case of an expressive security policy, the permissions are assigned indirectly to the user. For instance, in case of the Attribute-based Access Control (ABAC) [94], permissions are defined according to subject, object, and environmental attributes. These attributes may change over time to have different values entailing changes in the condition that permits an event. In consequence, verifying the compliance between a logged event and a security rule should not consist in checking the validity of the condition at the time of the investigation but rather at the time of the access.

Moreover, similarly to the condition, security rules might also change over time. These changes are usually controlled by an administrative policy that is managed by administrators. That being said, it is important to consider the variation of the policy over time, as well as monitoring administrative actions since administrators themselves can commit violations.

Last but not least, the a posteriori access control mechanism should be combined with a dissuasive sanction and reparation policy to deter the user from violating the security policy.

In this work, we provide novel means and solutions to solve these problems; hence, improve the a posteriori access control.

1.3 Contributions

We cover the three steps of the a posteriori access control that are log processing, log analysis, and accountability.

First of all, to ensure log processing, we propose a new method to extract useful information from logs in terms of the security policy. The goal of our solution is to resolve the heterogeneity between log formats in provenance of different log sources, while avoiding treatments on log files for the inconvenience that they provide in terms of memory consumption and processing time. Instead, we treat the queries that are sent to the logs by rewriting them using a semantic mediator. We show that this method is more efficient than traditional log processing methods as it guarantees scalability, system autonomy, and transparency in accessing data location and formats.

Moving on to log analysis, we get into different new aspects that leverage the a posteriori access control such as the semantic enrichment of logs, and the policy temporal compliance in both cases of a static and evolutive policy.
To have an effective detection mechanism, we need to compare between logged events and the security policy. Therefore, we treat the semantic enrichment of the extracted log information that aims to complement the logged data with complementary information relating to the conditions defined in security rules. This is useful in case of an expressive security policy, where a good number of attributes, that cannot be found in logs, are needed to define access permissions. Thus, we develop a multi-agent system architecture to automate the information collection process with respect to the security policy; hence, handle the semantic enrichment of logs.

Moreover, we improve the a posteriori access control by considering the temporal aspect when checking policy compliance. As the investigations are done a posteriori, we highlight that a correct policy conformance evaluation resides not only in respecting the required security attributes but also at the right time. To formalize this temporal verification, we use the Event Calculus (EC), a formal language for representing and reasoning about dynamic systems, that we implement in SWRL, and integrate in the multi-agent system.

Besides, we consider the policy temporal compliance in both cases of a static and an

evolutive security policy. By static policy, we mean that it is its expression that is static in the sense that it is defined once and for all, while its application is not static and may depend on contextual conditions. In the second case, we suppose that the rules defined in the security policy are subject to change over time using an administrative model of the policy. Therefore, the policy evaluation process starts by getting the rules that were in place at the time when an access occurred. Once again, we use the Event Calculus to express the relation between a logged event and the rules that held at the time of the access to detect violations, as well as the inter-dependency between administrative actions and the valid security rules. In addition, we consider the violations that can be caused by both users and administrators, which make the evaluation process recursive. Therefore, we define a termination condition.

Continuing, we define a framework for accountability in the a posteriori access control, to decide whether or not the user should be sanctioned once a violation is detected. We also consider the case where responsibility can be transferred to the administrator. To the best of our knowledge, our work is the first one to consider the temporal verification and the evolution of the security policy over time, as well as to propose an accountability framework when performing an a posteriori access control .

Finally, for each of the above contributions, we provide use cases in the healthcare domain where the a posteriori access control can be deployed. In addition, we present use cases that were provided from a real organization, and we argue how the quality of data can affect the a posteriori access control. We sum it up by exposing some issues to raise awareness among organizations in order to have a better post-access control.

1.4 Thesis Outline

The rest of this document is organized as described below.

Chapter 2 - State of The Art - reviews some concepts and works that are related to our field of research. We present different access control models, and we justify our choice of using the Attribute-Based Access Control model (ABAC). We also distinguish the difference between the a priori and the a posteriori access control, and provide a

literature review of the existing works relating to the a posteriori access control and compare them. Moreover, since the a posteriori access control is primarily based on logs and the security policy, we study some log processing and analysis methods, as well as some works treating the accountability problem in addition to policy representation models. Finally, we present the Semantic Web technologies and tools based on which we built our approach.

Chapter 3 - Extracting Log Information Using Semantic Mediation - presents the architecture of the semantic mediator that we used to rewrite log queries and extract information from logs in terms of the security policy. The idea is not to treat logs but rather to handle the queries that are sent to them. We decompose the query rewriting process into two steps: Semantic rewriting and Syntactic rewriting, and we show how this can be done by modeling the concepts present in logs using ontologies and establishing mappings between them. We also provide three scenarios that serve as use cases and present the used open source tools to implement the mediator.

Chapter 4 - A Posteriori Violation Detection with a Static Policy - handles the semantic enrichment process and the policy temporal compliance problem in case of a static policy. In this chapter, we consider that the expression of the security policy does not change over time, and we highlight that the importance of the a posteriori investigation resides not only in checking policy compliance but also in verifying the respect of the security rules at the right time. Thus, to enrich logged data, we provide a multi-agent system architecture to gather complementary information, and we model the verification process according to the Event Calculus and SWRL to reason over time, in addition to illustrating a use case.

Chapter 5 - A Posteriori Violation Detection with an Evolutive Policy - considers the evolution of the security policy over time that is triggered by an administrative policy. Therefore, monitoring administrative actions is also considered since each time we want to verify the legitimacy of an event, we need to get the rules that held at the time of its occurrence; hence, consult the administrative actions and check their validity. That being said, the verification process becomes recursive, the reason why we define

a termination condition. We express this inter-dependency between administrative actions and valid security rules using the Event Calculus and SWRL, and provide a use case as well.

Chapter 6 - Accountability in the A Posteriori Access Control - proposes an accountability framework for the a posteriori access control, and discusses how it can be seen as a requirement and as a mechanism. We suggest to integrate a *justification obligation* in the accountability process to increase the probability and the severity of sanctions, as not respecting this obligation is a violation by itself. Moreover, we treat accountability in both cases of a static and an administrative security policy and propose a modality to apply sanctions.

Chapter 7 - Conclusions and Perspectives - concludes this work by summarizing the contributions that were lead in this thesis, and suggests new perspectives to be treated in future researches. Moreover, it illustrates three log analysis use cases that were provided from a real organization. We show how the quality of the provided data can influence and even block performing the a posteriori access control. As a result, we open a discussion that provides certain recommendations about the information that should be present in logs as well as the conception and implementation of the security policy. These discussed issues need to be considered by organizations to be able to perform an a posteriori access control.

Chapter 2

State of The Art

2.1 A Priori vs A Posteriori Access Control

The main difference between the a priori and the a posteriori access control is how access permissions are decided. While in the a priori access control access permissions are independent of the user's experience, they are known a posteriori on the basis of the experience. The distinction between a priori and a posteriori access thus broadly corresponds to the distinction between empirical and non-empirical access. The choice of deploying an a priori or a posteriori access control usually depends on the environment. For instance, the a posteriori access control is more advantageous in environments (e.g., healthcare) where users need to go ahead with their duties, without worrying about access authorizations problems. Meanwhile, other environments require robust security guarantees (e.g., military information systems), making the a priori access control more suitable. Therefore, in many settings, the a posteriori access control cannot replace the a priori access control because the costs of incidental misuse are much higher than the costs of a preventive security mechanism. Although the balance between the security requirements and the availability of services is assured when deploying the a priori access control, it is important to consider the separation of concerns. As a consequence, the a priori and a posteriori access control are being unified, the case in which a "break-glass" mechanism (*c.f. Section 2.1.1*) is integrated to provide flexibility in access authorizations. Thus, relying on the a posteriori access control is motivated by the need of ensuring that users are not performing malicious actions when the "break-glass" is activated.

Still, this a priori/a posteriori nuance is always relevant to justification or warrant, that is, the security policy.

Access control policies are high-level requirements that specify how access is managed and who may access information under what circumstances. They are enforced through a mechanism that translates a user's access request, often in terms of a structure that a system provides [177]. Moreover, access policies can be represented according to several models that bridge the gap of abstraction between policy and mechanism. These security models are formal representations of the security policy that serve in describing the access security properties as well as providing theoretical limitations of a system.

In fact, these models can be used in both a priori and a posteriori access modes. While they are used for granting users access in the a priori mode, they are considered as a referrer that is consulted to detect violations in the a posteriori mode.

As access control has a large body in research, we will present, in the following, some expressive access control models which mechanisms are used to enforce the a priori access control, and then talk about the a posteriori mode of this latter.

2.1.1 A Priori Access Control

The a priori access control is the traditional access control in which users are prevented from gaining access to a resource outside their sphere of access. It is deployed to stop unwanted or unauthorized activity from occurring by blocking it immediately if it is not explicitly allowed in the security policy. Thus, access privileges are resolved before granting access to users.

Ideally, access control models come into the picture to define the right level of permission to be granted to an individual to perform his duties; hence, we present three expressive access control models.

Role-Based Access Control

The Role-Based Access Control model (RBAC) [70], was introduced in 1992. It makes it possible to establish access control over the applications and services within a

company based on the definition of the roles to be assigned to users and resources. Therefore, access to objects is based on the user's role and rules indicating which access is allowed for which given role.

We distinguish two types of relationships that lead to the relation user-permission, that are, the user-role assignment and the role-permission assignment relationships. The former present the roles that are associated to each user, and the latter assigns permissions to each role.

Furthermore, the implementation of the basic functionality of RBAC is called *Flat RBAC* where users obtain the permissions they need by acquiring the required roles. It is worth mentioning that there may be as many roles and permissions as the company needs. An abstract illustration of RBAC is shown is *Figure 2.1*.

In contrast, RBAC is not a linear monolithic model, since relationships may exist

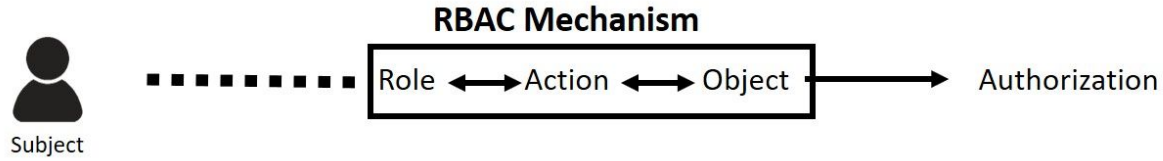


Figure 2.1: RBAC Mechanism.

among the roles themselves. This is where the *Hierarchical RBAC* comes on board in which higher-level roles subsume permissions owned by sub-roles.

Nevertheless, when the number of roles increases in a company, the number of roles in RBAC increases subsequently to properly encapsulate the permissions, leading to what is called *role explosion* that can become a complex affair. In addition, it is not adapted to a dynamic and distributed context as it assigns roles statically to the user.

Organization-Based Access Control

The Organization-Based Access Control (OrBAC) [62] is an access control model that appeared in 2003, in which the expression of an authorization policy focuses on the concept of organization.

In contrast, access control models are usually based on three entities: subject, action,

and object. Thus, to control access, one specifies whether a subject has permission to perform an action on an object. Since the main purpose of OrBAC is to allow defining a security policy independently of its implementation, it introduces new entities as an abstraction level for each of the subject, object, and action, that are the role, the view, and the activity, respectively. These latter are defined relatively to an organization as shown is *Figure 2.2*. Therefore, in OrBAC, the security policy specification is completely

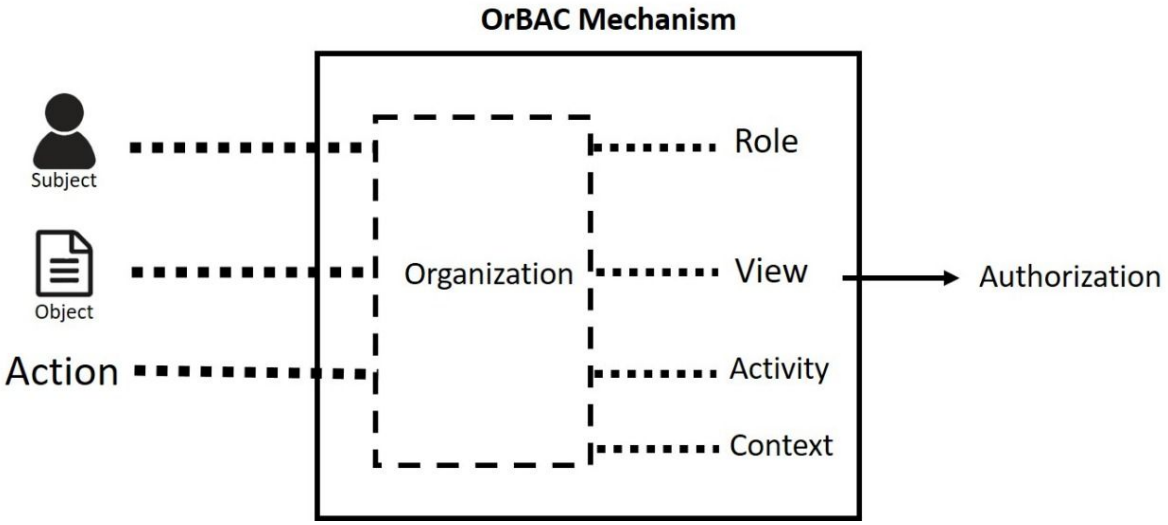


Figure 2.2: OrBAC Mechanism.

set by the organization. It is also possible to specify simultaneously several security policies associated with different organizations.
It must be pointed out that OrBAC is not restricted to permissions, and also includes the possibility to specify prohibitions and obligations. Moreover, from the three abstract entities (roles, activities, views), abstract privileges can be defined, and one can derive later on these abstract privileges in order to obtain concrete ones.

Furthermore, OrBAC has the notion of context, so its security policies can be expressed dynamically in addition to its support of hierarchy (organization, role, activity, view, context).

Attribute-Based Access Control

The Attribute-Based Access Control model [94] defines an access control paradigm in which access rights are granted to users through the use of rules that combine

attributes. Thus, access policies can use any type of attributes such as subject attributes, object attributes, environment attributes, etc., as shown is *Figure 2.3*. These attributes contain information given by a *name-value* pair. We present the main ABAC entities as follows:

- **Subject Attributes:** Subjects are the entities that request to perform operations on objects. Each subject can have one or more attributes, and is usually assimilated to a user. Such attributes may include the subject's name, role, affiliation, address, etc. Interestingly, the use of subject attributes makes access control lists (ACLs) and RBAC particular cases of ABAC where "identity" and "role" are respectively considered as attributes.

- **Object Attributes:** Objects are the requested resources to be accessed by the subjects. Similarly to the subjects, they have attributes that are important to make access control decisions. Object attributes are useful to specify the type of operations that can be done on the objects (e.g., read a document, excute a program, etc.). Examples of object attributes can be the type, location, owner, etc.

- **Environment Attributes:** *also known as Context Attributes*. They are the operational or situational context in which accesses are done, and are independent of subject or object. Environment attributes may include the time, location or dynamic aspects of the access control scenario, etc.

This aspect of using multiple attributes makes ABAC a flexible and multi-dimensional access control system that is capable of supporting any access control model. Therefore, its support of making fine-grained access decisions made it successful as it represents a rich policy specification and any number of attributes can be added within the same extensible framework. However, one drawback of ABAC is that it is hard to configure due to the way policies must be specified and maintained.

Limitation of the A Priori Access Control

Once the access control model and policies are set up, the underlying access control mechanisms will ensure that they are enforced in regular operation. However,

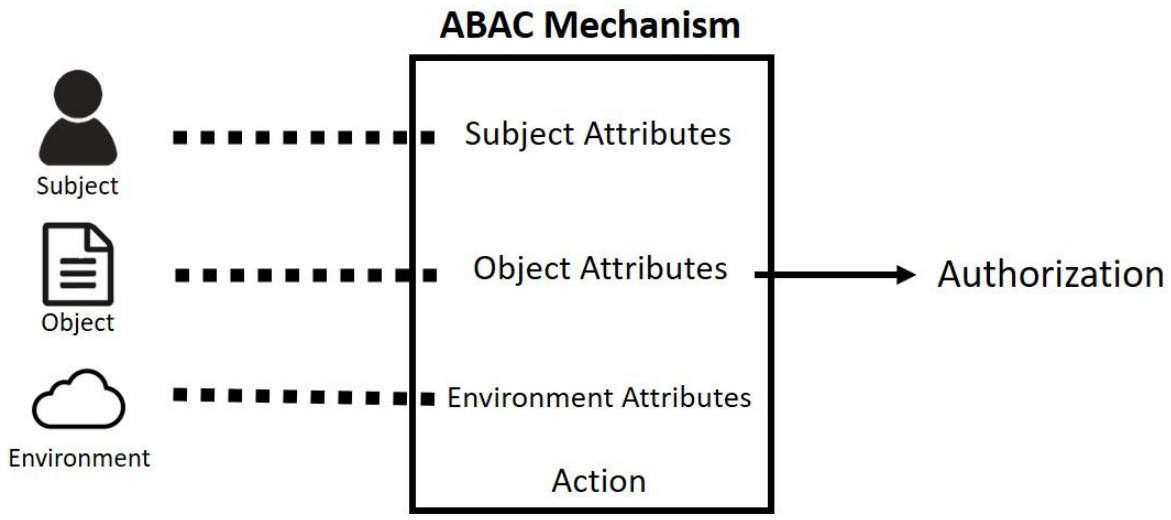


Figure 2.3: ABAC Mechanism.

implementing the a priori access control is insufficient to assure the desired security level since it is impossible to anticipate all usage scenarios when setting forth the policies. This problem of the inability of traditional access control models to handle exceptional situations has been known for more than twenty years [26]. Although risk-based models [41, 146], that weight the risk of granting access against the perceived benefit, were proposed to adapt to dynamic environments, it is still difficult to distinguish between the malicious break-in and well-intentioned infringements as access decisions are made in real-time. To address this problem, additional mechanisms should be deployed on top of the underlying access control models to increase flexibility while maintaining a certain security level at the same time.

One common strategy is the "break-glass" [71, 131, 173], which has been introduced to handle emergency situations by breaking or overriding the standard access permissions in a controlled manner. It is derived from the action of breaking the glass and ringing the fire alarm [181], and refers to a quick means for a person who does not have access privileges to certain information to gain access when necessary. This principle was originally brought for disaster management [157], and is usually implemented by issuing temporary accounts that comprise more powerful access rights on one hand, and more detailed logging on the other hand. In this respect, it allows a subject to act under certain conditions even though he/she was

not previously authorized to do so. Moreover, it is usually implemented in an ad-hoc manner during the administration phase. As it might complicates the a priori analysis of the security policy, it is crucial to perform monitoring to assure the control of the separation of the regular policy and the emergency mode. Therefore, a variety of different approaches were published, offering different features for different application domains. For instance, [166] proposed an optimistic access control scheme as a paradigm for constraining access in unexpected situations. In their approach, the authors assumed that most accesses would be legitimate, and the preservation of the organization's security is ensured by external controls. Moreover, traceability was provided using monitoring and recording functions that are based on the Clark-Wilson Integrity Model [43]. Furthermore, [7] presented an exception-based access control solution to handle "break-glass" attempts in healthcare systems. In case of an emergency, the solution permits policy override if no emergency policy exists, while notifying the administrator of the respective override. Both of these approaches are burdensome for administrators as the enforcement of security policies relies on the capability of administrators to identify inappropriate accesses.

Besides, a break-glass extension for SecureUML was introduced in [30]. This extension supports model-driven development techniques based on role-based access control policies with break-glass, and the resulting SecureUML break-glass policies can be transformed into XACML [81]. The approach also integrates means for monitoring and logging the usage of emergency rights using obligations. [131] presented a break-glass model, named Rumpole, that takes into account the notion of subjects' competences and empowerments to gain more insight into the causes for the access denial, rather than on a set of emergencies or explicit override permissions. Other studies were interested in integrating the concept of break-glass policies into the business process context [149, 172, 204].

In all cases, the "break-glass" mechanism should be invoked in conjunction with a strict accountability function that offers both logging and auditing, of which the users should be aware of in order to be discouraged from abusing the regular permissions beyond emergencies. Such functionalities are the core components of the a posteriori access control which we discuss next.

2.1.2 A Posteriori Access Control

The a posteriori access control can be classified as a deterrent and detective access control at the same time [195]. It is deterrent because it picks up where prevention leaves off. It allows access without imposing prevention constraints, which may lead to possible or successful attempts of violation. Moreover, the a posteriori access control deploys a monitoring mechanism to detect illegitimate accesses; hence, it is detective since the inspections are applied *after-the-fact* rather than in real-time.

In order to not block users in particular cases (e.g., emergency) [157], and to allow them to access the resource they need, a "break-glass" mode is adopted which leaves the access relatively "open". Thus, the user is allowed to override the access restriction voluntarily, with or without the intervention of the administrator. Nevertheless, these accesses are traced and audited afterward.

Although in legal terms the "harm is constituted", these accesses are made with full knowledge of the user of the potential consequences. He has been warned previously and yet, still performed these accesses without a delegation. Moreover, if the user could not prove the legitimacy of his actions after being questioned, he is held responsible and should be sanctioned by enforcing the fact that he was warned.

In contrast, decisions are taken by consulting the security policy that usually defines the access requests that must be authorized. On the other hand, log files record "What happened? When did it happen? And by whom" in the system. Therefore, it is referenced for diagnostic, audit trail, and investigative purposes in case of malicious activities, system attacks, or security breaches [152]. It can also be used for accounting purposes as it is obviously a trusted source that offers accurate data when considering that all accesses are logged. Therefore, logs constitute the basis on which the a posteriori access control lies to proceed in the investigations and provide proofs, and the security policy takes the role of the judge as it defines the rules that permit accesses. We can thus designate the a posteriori access control as an alliance between the logs and the security policy.

After what has been discussed, the a posteriori access control was defined in the literature as composed of three stages: *log processing*, *log analysis*, and *accountability* [38,

48, 10]. The first step is meant to extract relevant information from logs. This latter is then analyzed in the second step to determine whether there has been a violation or not. Thus, the investigator can obtain evidence demonstrating that the user did not violate the security policy and that the problem was caused by external malice or a system error, or that there was effectively a violation. When abuse of privilege is assumed, the circumstances under which the user performed such action are studied to see if the access was harmful. Finally, the last step is about assigning responsibility by applying sanctions and remedies once the violation decision is made.

This coupling between logging and auditing attends to assimilate the a posteriori access control to intrusion detection. Although these two resemble, they are still not the same. So what is the difference?

Difference between the A Posteriori Access Control and Intrusion Detection

As the volume of logged activity increases quickly, automated tools are being developed and used to reduce human tasks and help in carrying out auditing. One class of these tools is known as Intrusion Detection systems [58], which can be classified as passive or active.

Passive systems [124] analyze the audit data offline and bring possible intrusions or violations to the attention of the auditor who then takes appropriate actions, while active systems [96] analyze audit data in real-time. In addition to alerting the auditor of the violations, these systems may take immediate protective response on the system. However, when the detection is passive, the distinction between the a posteriori access control and intrusion detection can be confusing as in both cases the goal is to detect violations.

Intrusion detection systems can be based on different approaches such as Threshold-Based approach [161], Anomaly-Based approach [112], Rule-Based approach [97], etc. Each approach is used to detect a specific type of violation. For instance, threshold-based systems define abnormal use concerning pre-specified acceptable thresholds (e.g., number of login attempts), while anomaly-based systems define abnormal use as a use that is significantly different from what normally observed (e.g., statistical measures). As for the Rule-based approach, it describes what

is suspicious based on known past intrusions. It is generally enforced by the encoded knowledge of security experts of properties characterizing past intrusion events.

It is remarkable that whatever the used approach is, intrusion detection systems depend on the expertise of the person who is developing them. Most of the time, they can only detect violations that involve anomalous use defined following what the general behavior should be. Therefore, as an attacker can always penetrate the system by employing new techniques, machine learning approaches are being developed to be able of autonomously learning new attacks [103, 34]. Nevertheless, they tend to learn to detect certain patterns and are often dependent on the system's response or feedback.
In this regard, the a posteriori access control cannot be categorized as passive intrusion detection since, as mentioned earlier, a trustworthy environment is considered where the main concern is not to discover any violation, but to detect the potential ones of the security policy. As access permissions may evolve depending on circumstances, an intrusion detection system might consider a legitimate access as abnormal. Thus, it is the security policy that manages the access control requirements fulfilling the objective of not preventing attacks, like in the intrusion detection, but rather fixing responsibilities and applying sanctions.

Previous works on the A Posteriori Access Control

The motivation of using an a posteriori access control model was brought with the difficulty of managing access control in many environments and organizations. However, in certain environments, it is important to assure the continuity of daily activity services, where sensitive fields are involved. This might be the case for some medical organizations, where several emergency situations may occur.

In an a posteriori access control model, a trust management system is used to ensure that data resources are only provided to users who are subject to penalties in case of violation. This auditing process is conducted using audit proofs such as logs. A number of researches dealt with this type of access control, which we present in the following.

One of the first works to address this problem was [48], where the authors proposed

a language that allows agents to distribute data with usage policies in a decentralized architecture. They designed a logic that allows agents, who can be audited at any time, to prove their actions and authorization to possess particular data. The proofs of the users rely on a usage policy that is attached to the data and which contains a logical specification of which actions are allowed to be done on the data and under which condition it can be redistributed. Moreover, they showed how this logic allows different kinds of accountability (agent accountability and data accountability), and demonstrated the soundness of this logic. This work was then extended in [38], to include the ability to specify conditions and obligations within the policies, by allowing the agents to refine the policies before passing them to other agents. In contrast, as in [48] the only allowed policies were those that are explicitly stated by the data owner, [38] introduced three new functions that are: observability, conclusion derivation, and proof obligation. Besides, the authors formalized their proof system in the proof checker Twelf [163], that allowed them to model proofs provided by agents, and the subsequent checking by the authority. Continuously, a proof finder that allows agents to generate valid justification proofs was implemented in [36].

[67] also introduced the a posteriori access control, and provided a logical framework for a posteriori policy enforcement that combines trust management and elements of audit logic, called APPLE. In this framework, users are responsible of logging and keeping traces of their actions, and each data item is governed by its own policy label. Moreover, they considered that the log is secure in the sense that users can log actions, but cannot modify an entry in the log, and used trusted components to assure that all communications are logged. They also focused on monitoring the transmission of documents, considering that the gravest policy violations can occur through this latter. This approach is less specific with regard to the expressive power of the policy rules, but it is more precise with regard to how the policies appear in the system, namely as sticky policies attached to the data items.

The above works [48, 67] were the first of their kind; hence, it is evident that they have some drawbacks. First of all, the responsibility of logging and monitoring actions is imposed on the users who are performing them. This is however not realistic even when considering that the users are trusted. There should be a policy that defines the perimeter of the logging process to control the flow of this latter. Next, security

policies are attached to the data without being defined according to a security model. Moreover, they did not provide the log information collection or log querying process.

On the other hand, the a posteriori access control had a wide success in the healthcare domain. For instance, [59] outlined the needed architecture to apply audit-based access control in electronic health record systems, and discussed the advantages and limitations of their proposal. Other efforts in the medical domain were [12] and [11].
In [12], the authors proposed to restructure IHE-ATNA log records [10] according to an OrBAC security policy model to find policy violations. The core idea was to structure these logs to bring them close to the security policy by using a reformatting procedure that maps the relevant structures and contents of logs to the concepts of the used policy. Contrary to previous works that mainly focus on security languages, the proposed security control process is based on a contextual security model having an appropriate level of abstraction in addition to the ability to converge logging data and policy structural concepts. Furthermore, in [11], they defined and enforced the extraction of necessary data from logs, for policy violation detection, by building an ontology model of these logs and querying it. While they adopted the ATNA standard as a log format, it is possible to consider other log formats [84].

We can consider [12, 11] as best efforts for taking into account an expressive security policy when performing the a posteriori access control. However, they did not treat the accountability process as well as the modifications of the security policy.

Other applications of the a posteriori access control can be also related to business processes such as [8] that provided an approach allowing modeling an auditable process by using Business Process Management Notation (BPMN). They showed how security policies of a business process can be expressed using BPMN models, and provided an example from the banking context to illustrate an auditable process. Moreover, the a posteriori access control was used for detecting violations of privacy protection rules in social networks [14], as well as in usage control. Usage Control [158] includes obligations that are mandatory requirements that a subject has to perform after obtaining or exercising rights on an object. Although the objectives of the a posteriori access and usage control are the same, there is no guarantee that after

granting access to a resource, the user will fulfill the imposed obligation. Thus, it is necessary to control its risk exposure. For instance, [15] proposed a trust-and-obligation based framework that reduces the risk exposure of an organization associated with a posteriori obligations. Their methodology is based on evaluating the access requests that trigger a posteriori obligations and checking the requesting users' trust values to decide if they can fulfill their obligations. Besides, their framework detects and mitigates insider attacks, and unintentional damages that may result from violating a posteriori obligations, in addition to determining misconfigurations of obligation policies.

A summary of the above discussed works is presented in *Table 2.1*. As we can see, the aforementioned works have their own limitations. Thus, we fill the gap by improving certain aspects and addressing some new ones that were not treated before, such as policy temporal compliance, policy changes, and accountability.

As stated earlier, the a posteriori access control cannot be done without two fundamental concepts that are logs and the security policy. For accountability purposes, useful events should be extracted from logs, and security rules should be defined to quantify the violation level. Therefore, we present, in the following, a literature review about each component of the a posteriori access control and access policies representations.

2.2 A Posteriori Access Control Steps

None of the previous works on the a posteriori access control that we presented in *Section 2.1.2* provided a complete solution that covers all the areas of the a posteriori access control that are log processing, log analysis, and accountability. [12, 11] are the only ones to provide a log information extraction module, as well as means to analyse logs in terms of the security policy. As for accountability, the only one that treated the problem is [48]. However, their vision of accountability is different from ours, as for them an agent passes the accountability test if he provides proofs that rely on a usage policy that is attached to the data and that specifies which actions can be done to this data. In our work, we consider that the user should justify his

Table 2.1: Comparison between previous works on the a posteriori access control

Work	Application Domain	Proofs & Logging	Monitoring	Policy Specifications	Penalty Mechanism
JG Cederquist et al. [48, 38, 36]	Decentralized systems & Medical domain	Ensured by users	Conducted by auditing authorities	- Usage policies attached to each data and created by the data owner - Logic representation with no security model	Not applicable
S. Etalle et al. [67]	Collaborative environments	Ensured by users	Conducted by auditing authorities	- Sticky policies presented as documents' labels - Logic representation with no security model	Not applicable
H. Azkia et al. [12, 11]	Medical domain	Ensured by an ontogy-based log filtering module	Conducted by the system's workflow and the deployed security policy	- Logic representation according to the OrBAC model	Not applicable
M. Karim Aroua et al. [8]	Business processes	Ensured by users	Conducted by auditing authorities	- Usage policies attached to each data and created by the data owner - Logic representation with no security model	Not applicable
L. Bahri et al. [14]	Social Networks	Ensured by a Bitcoin chain attached to the object	Conducted by auditing authorities	- Privacy rule-based representation according to the Discretionary Access Control Model (DAC)	Not applicable

actions during the accountability process and could be subject to sanctions based on his justification. Therefore, we present in the following some works relating to each step of the a posteriori access control (log processing, log analysis, and accountability) in general.

2.2.1 Log Processing

To have a correct log analysis, pertinent information should be extracted from log files. One problem to be addressed when analyzing logs, is the multiple log file formats that are due to the variety of log sources. These logs keep traces of all the established events in the information system, and these events differ from one logging source to another. Moreover, sometimes a log format may differ between the versions of the same source. For instance, the format and content of an application log are usually determined by the developer of the software program.

The key point in log analysis is that it needs to interpret messages within the context of an application or system, and map varying terminologies from log sources [3]. It turns them then into a uniform terminology to produce clear reports and statistics.

While it is acceptable to keep the log formats as "they are" in some cases, the a posteriori access control requires having log information in terms of the security policy to facilitate the violation detection, as mentioned earlier. Therefore, one might think that to be able to converge log formats and the security policy, it is required to transform these various formats into some common format. This process of unifying the different logs is called "log normalization". Several efforts have been made to standardize log formats and make normalization obsolete [193, 46, 98]. Unfortunately, these efforts added variations to the format instead of reducing it, allowing event normalization on very different levels of detail. Therefore, the necessity of dealing with the differences of formats persisted, and further processing to obtain relevant information is still needed. This was the core motivation behind log normalization. In this respect, we review in the following some common log normalization methods that can be observed on the market and the research community.

Popular Log Normalization Methods

Regular Expressions The use of regular expressions (regex) is one of the classical approaches of log normalization. It is based on extracting data from log sources, by applying several regexes until a match is found, that is then transformed to the normalized output data [74]. In practice, regexes' runtime requirements make them unsuitable for large classes of applications, especially in enterprise environments with a large log volume. To solve this problem, two different approaches were proposed: the traditional one is to hardcode parsers for each log format that is very important or requested by a user, and the alternative is to use advanced data structures suitable for fast parsing. However, their processing remains intensive and wasteful in terms of memory.

Tokenization In this type of normalization, an actual log event is split up into fragments that contain granular, yet useful data called tokens. The simpler case is comprised of white space splitting. However in the case of logs, tokens may be different. For instance, some tokens may be words, but other tokens could be symbols, timestamps, numbers, or any particular notation that is present in the log [179]. A famous implementation for tokenization is Apache Lucene [6] that groups log events containing the same words. One drawback of this method is that it heavily relies on static terms present in the logs.

Natural Language Processing (NLP) Natural Language Processing (NLP) is a branch of Machine Learning that offers the ability to a program to understand human language. The goal of this normalization method is to use NLP techniques to reveal a structure of the system logs that is similar to natural language. In consequence, log records become human-readable as subjects, objects, verbs, and more are identified. Nevertheless, NLP cannot be defined in a general way since many different techniques can be used throughout the process such as Conditional Random Fields (CRF) [120, 183] that is about segmenting or labeling sequential word data and that is usually used for chunking speech texts, Word Embedding [82] which produces a mapping from the set of words of a text corpus to an euclidean space, N-gram frequencies extraction [35]

that extracts a contiguous sequence of n items (words) from a given sequence of text or speech, and TF-IDF [109] that considers the frequency and the discriminative power of a word in a document. We refer to [117, 23, 205, 188, 211] as concrete examples of using this technique for log analysis.

Custom Normalization Custom normalization is the most effective yet the most complex traditional normalization method. For each log format, a different filter and parser are used. For instance, a CSV parser is used for CSV logs. This type of normalization can be observed in some log analysis tools such as Logstash [127] in which the *GROK* (Graphical Representation of Knowledge) filter is commonly used. Logstash searches for the specified GROK patterns in the input logs and extracts the matching lines from the logs. The output is a more structured data which makes it easy to search and to perform queries.

The above normalization methods are usually used to get information from logs to provide evidence of unusual behavior when performing log analysis. Although a lot of common techniques, built upon these methods, are used to analyze logs such as filtering and finding patterns [89], they are not as effective as they were before due to the growth of the generated logs and the need to identify the correlation between log events. Conversely, the use of the Semantic Web Technologies, notably, ontologies (c.f. *Section 2.4*), showed itself in the context of security log analysis as a possibility of improving the results of searching in log files. Therefore, many works used ontologies as a guide for extracting rules or patterns, making it possible to discriminate data by their semantic value and thus, extract more relevant knowledge. We present some of these works in the following section.

Normalizing Logs with Ontologies

An effective log analysis impose the need of having structured information. Thus, a number of approaches focused on a conceptual formalization of logs that is properly built into the application context to provide a better security assessment. Ontologies can be used as a vocabulary, a dictionary, or a roadmap of a particular domain.

Furthermore, an ontology can be used to provide inferences about relationships between entities. More details about ontologies are given in *Section 2.4*. That being said, ontologies are being more employed to process and analyze logs for the advantages that they offer. For instance, [190] showed how modelling logs with an ontology can make it easier to find correlation between event logs. The authors focused on the logs of web application firewalls, and modelled their domain by identifying their major classes and relationships manually. Their scope was narrowed to ontology engineering rather than their processing. In [147], a framework that can process any log file and automatically generate a semantic interpretation using RDF linked data triples is proposed. It begins by normalizing the log into columns and rows using regular expression-based and dictionary-based classifiers. The obtained entities are then mapped to concepts in general knowledge-bases (e.g., DBpedia), as well as domain specific ones (e.g., Unified Cybersecurity Ontology). Therefore, cell values are linked to known type instances (e.g., an IP address), and relationships between columns are deduced. The authors also showed how converting log files into such semantic representations reveals their meaning and supports search, integration, and reasoning over the data. Other ontology-driven log representations are [174, 153].

After what has been discussed in this section, we can assimilate modelling logs with ontologies to another normalization method for its intention to unify log formats. Nevertheless, although it is more advantageous than the classical methods regarding the addition of semantics and the power of reasoning, it remains very consuming in terms of time and processing, since logs are in a constant growth. Therefore, we attempt to overcome this problem while incorporating the advantages of ontologies. For this concern, we use semantic mediation techniques to extract information from logs and analyze them more efficiently. More details are presented in *Chapter 3*.

2.2.2 Log Analysis

Log analysis may be the most under-appreciated, unattractive aspect of information security. However, it is one of the most critical security processes since it helps in understanding and making sense of the generated log records in an application domain. This process helps the organizations in maintaining their security on different

levels, such as security policies' compliance, audits and regulations, troubleshooting, and understanding users' behaviors. As SANS Institute puts it, *"Logging can be a security administrator's best friend. It's like an administrative partner that is always at work, never complains, never gets tired, and is always on top of things. If properly instructed, this partner can provide the time and place of every event that has occurred in your network or system"* [76].

In the case of the a posteriori access control, log analysis is meant to check the conformity of the deployed access policy, by detecting suspicious activities that deviate from security rules, and repairing anomalies if any. Therefore, organizations must analyze and review their logs from one time to another to assure that the security policy is being respected. In the following, we review some techniques that were previously used for log analysis.

Machine learning techniques for log analysis

Log analysis is the core component of the a posteriori access control to assess policy violations. As they become incredibly famous, machine learning techniques are being widely used for automating the detection of abnormal logs. We can distinguish different types of these techniques such as classification, clustering, and statistical techniques [91]. Classification is a kind of supervised learning techniques which means it needs labeled data to supervise the learning process. It has been previously used for anomaly detection [16]. Despite the advantages that it provides in terms of correct answers during the training, its inconvenience resides in the difficulty of getting enough reliable labeled data. In fact, it is hard to provide absolutely problem-free data especially when dealing with logs, and it is always possible to label some erroneous data as normal which can lead to false positive errors. Moreover, supervised learning will fail to detect a totally unknown problem. Moving on to clustering, it was also used for log analysis in [148], where the authors presented a case study about using Self-Organizing Feature Maps (SOFM) [118] on the server log data. However, they only considered numerical data. Even if they showed the advantage of SOFM for being independent of the data distribution and cluster structure, its major shortcoming is that the resulting clusters depend on the metric over which the clustering algorithm will

reason, which may leave certain anomalies undetected. As for statistical techniques, [211] used the Principal component analysis (PCA) [162] to detect anomalous logs in the area of log analysis. Although this method is effective when dealing with high-dimensional data, the disadvantage of using statistical techniques for log analysis is that they need a threshold to separate the normal data from the abnormal one, and selecting a suitable threshold is hard and needs prior knowledge.

As it has been discussed, each machine learning method has its disadvantage when addressing log analysis. Moreover, even if they are being improved for a more accurate analysis [126], the result consists in detecting outliers of the general behavior rather than violations of the security policy. Therefore, as we want to treat the semantic enrichment of logs to converge them with the security policy, we study works that are related to the semantic log analysis.

Semantic log analysis

The use of ontologies proved itself to be advantageous in the area of log analysis for the semantic value that they provide and their capability of capturing the structure of a domain and its possible restrictions. In this connection, many researches treated the problem of log analysis by addressing the semantic enrichment of logs. For instance, the authors in [61] proposed a platform for semantic security log analysis. The platform aims to reduce the manual work of linking information from disparate log sources, by contextualizing the different information in an ontology, providing analysts a common vocabulary to query logs. Moreover, [116] enriches event logs by integrating them with other organizational data sources and mapping both of them to the TOVE Ontology (TOronto Virtual Enterprise). After this integration, reasoning over the ontology is applied to answer questions. One drawback of this approach is that in case of an update in a data source all the ontology axioms must be recomputed. Another thing is that they consider the enrichment process before extracting information from logs, in other words, while querying the logs. Another work related to log semantic enrichment is [156]. It proposes an Emergency Response (ER) log management application, where ER log files are processed and enriched with semantic metadata by means of information extraction. The extraction process applies

a knowledge-intensive approach. For example, gazetteers were deployed to match place names and first names with the log text, and available web-services were used for semantic annotations of text. However, this approach can only extract information that is as exactly defined in the adopted thesaurus, and does not consider the possibility of having different vocabularies. Moreover, it considers that all the needed information is logged (e.g., location), which is not always the case.
Other works analyze logs by detecting patterns. For example, in [150], the authors proposed a framework to semantically enrich web logs by structuring the contained information (users actions) using ontologies and applying mining techniques to detect patterns. In addition, the authors of [72] introduced an approach to automate the discovery of Workflow Activity Patterns in business processes by means of reasoning over an ontology modelling these patterns. To detect if a given pattern is present in a process, ontology individuals are generated automatically from event logs, and the semantics and sequence of events are being considered.

Since our goal in this thesis is to detect violations of the security policy, our semantic enrichment process will be about contextualizing the extracted information from logs with complementary information that is related to the security policy. In this way, the comparison between the logged events and the rules defined in the security policy will be possible, leading to decisive results about policy violation. This idea is presented in details in *Chapter 4*.

2.2.3 Accountability

Accountability in the a posteriori access control is about fixing responsibilities and applying sanctions and reparations if needed. As the accountability problem can be treated in different fields, it can be viewed through different angles, as each one has its own understanding of accountability.

For instance, the authors in [101] proposed an operational model for accountability-based distributed systems. They described analyses which support both the design of accountability systems and the validation of auditors for finitary accountability systems. Moreover, they explored the tradeoffs underlying the design of accountability systems including: the power of the auditor, the efficiency of the audit protocol, the

requirements placed on the agents, and the requirements placed on the communication infrastructure.
Furthermore, in [78], the authors discussed the issue of responsibilities related to the fulfillment and the violation of obligations. They proposed formal definitions of the different aspects of responsibility, namely causal responsibility, functional responsibility, liability as well as sanctions, and examined how delegation influences these concepts. Nevertheless, in our approach, we consider that a user is responsible a posteriori for committing violations. This is different from the concepts provided in [78], where responsibility implies the a priori obligation of accomplishing a task.
Besides, [69] built a responsibility model based on the concepts of Accountability, Capability and Commitment, which they developed using an UML class diagram. The model's objectives were firstly to help organizations for verifying the organizational structure and detecting policy problems and inconsistency. However, they did not provide a formal representation of the model.

In contrast, [21] provided an abstract language for accountability clauses representation (AAL) with temporal logic semantics. They considered that an accountability clause is a triplet (*uc, aa, rc*), where *uc* represents the usage control, *aa* is the audit that observed the violation of the usage control, and *rc* is the rectification to be applied. Therefore, we take into account these three components so our accountability mechanism could be easily expressed in AAL.

In *Chapter 6*, we discuss our vision of accountability, and we propose a mechanism that could be appropriate to the a posteriori access control.

2.3 Policy Representations

As it has been shown in *Section 2.1.1*, the security policy can be formally represented according to different access control models, as well as their extensions and variants [25, 129, 139, 114, 1]. This formalization allows the proof of properties on the security provided by the access control system being designed. However, given the complexity and the scope in which the definition of the security policies is involved, it is essential to have a framework to reason about these policies. It is fundamental to unify the

interpretation of access policies throughout the organization to make the whole system simpler and less error-prone. This is particularly important in the a posteriori access control where users' rights may need to change in order to cope with specific contexts such as emergencies.

A lot of researchers have spent a few decades focusing on the representation of policies and policy rules. For instance, the eXtensible Access Control Markup Language (XACML) [81] and the Security Assertion Markup Language (SAML) [88] are XML based frameworks that enforce access control based on attributes.

XACML was defined by OASIS, and it includes languages for expressing authorization rules and for access decisions. It has been known as a key standard that implements ABAC since its language specifies access control requirements using rules, policies, and policy sets, expressed in terms of subject (user), resource, action (operation), and environmental attributes and a set of algorithms for combining policies and rules. Therefore, attributes are a very important part in XACML and are evaluated by the rules in order to determine whether some restriction is applicable or not. Moreover, XACML proposes an authorization system that consists of five conceptual units: the Policy Enforcement Point (PEP), the Policy Decision Point (PDP), the Policy Administration Point (PAP), the Policy Information Point (PIP), and the Context Handler (CH). PEP performs access control by requesting an access decision to PDP, which uses the policies made available to it by PAP and the additional attributes sent by PIP to render its decision. The PEP communicates with the PDP and PIP through the CH that is an adapter between the XACML components and the protected application. As for the rules, they are integral to the functioning of XACML and form the core element in the hierarchy to make access decisions. A rule has target information, an effect, and a condition. The target is formed of a set of conditions that must be fulfilled to apply a policy or a rule on a given request. The rule's effect is to deny or permit access. The condition is optional, and its role is to refine the applicability of the target. Finally, rules must be part of a policy and can be evaluated separately.

SAML, also defined by OASIS, simplifies federated authentication and authorization processes for users, Identity Providers and Service Providers. It offers a solution that allows the Identity Provider and Service Providers to exist separately from each other, thus centralizing user management and providing access to SaaS solutions. The XML document that the Identity Provider sends to the Service Provider containing the user authorization is called SAML assertion, and has three different types: authentication assertion, attribution assertion, and authorization decision assertion. The first one proves the identification of the user and indicates the time the user logged on and the authentication method used, the second transmits the SAML attributes that provide information about the user to the service provider, and the third indicates whether the user is authorized to use the service.

Another flexible, extensible, and adaptable to a wide range of policy management requirements language is Ponder [55]. It is an object oriented language for specifying security and management policies presented as rules defining behavioral choices. Moreover, it allows the definition of positive and negative authorization policies, information filtering, and a simple delegation model. Key concepts of the language include roles to group policies relating to a position in an organization, relationships to define interactions between roles and management structures to define a configuration of roles and relationships pertaining to an organisational unit such as a department.

However, one common drawback of the above policy languages is that they do not consider the semantics of the policies. These latter are needed to fill out the policy's own framework, to make access control conditions predictable and interoperable, even where there is no prior agreement on the semantics of the access control conditions.
As a consequence, the motivation became to have a policy representation that generally relies on the expressivity of Description Logics (DL) [13], and particularly on OWL [133], for capturing the various knowledge artefacts that underpin the definition of a policy.

The use of OWL to define policies has several very important advantages. First, it is adequately representational to capture distinct activities that are required (obligations), restricted (prohibitions), and authorized but not necessarily expected (permissions), by an entity (subject) on a resource (object) within the system, and the circumstances

within which it applies. Next, the power of reasoning helps in determining access decisions and supporting analysis in case of policy conflicts.

A number of relevant approaches have been proposed in the literature that use OWL for representing access policies. For instance, in [196], KAoS, a multi-layer policy framework which supports policies described in OWL, was proposed. Monitoring and enforcing policy are done automatically based on OWL ontologies, and access rights are associated with different credentials and properties of entities. As for the Rei [113] policy language, it provides an ontological abstraction for the representation of a set of desirable behaviors by using flexible constructs like policy objects, meta policies, and speech acts to express different types of policies.

Furthermore, ROWLBAC [73] was also an effort to bring formalism into policy languages by modeling RBAC in OWL. Using OWL hierarchies and properties, different ontologies have been suggested, and modelling ABAC with OWL was discussed briefly. In [186], the authors showed how the Attribute Based Access Control can be represented in OWL, providing each one of the Discretionary Access Control (DAC) [142], Mandatory Access Control (MAC) [19], and RBAC according to the ABAC model. This latter work confirms that any model can be represented as an attribute based access control model; the reason why we chose to model the security policy according to ABAC in this thesis. Thus, we adapt the propositions of [73] and [186] to construct an ABAC policy ontology. Details about this ontology are given in *Section 4.3*.

Now that we have demonstrated the power of using ontologies in both modelling logs and the security policy, we provide some background on the Semantic Web technologies that we used to construct our solutions.

2.4 Semantic Web Technologies

The Semantic Web, also known as the Linked Data Web or the Web of Data, represents the major evolution in connecting information. It permits to integrate and combine data drawn from diverse sources; hence, to link the data from a source

to an other source. As its name implies, the Semantic Web is concerned with the meaning of the data more than its structure. In this respect, the data is machine-understandable which allows the computers to complete sophisticated tasks on behalf of the human. Moreover, Semantic Web technologies enable the creation of data vocabularies, querying the data, and writing rules to reason over this latter.

In computing and information science, vocabularies that represent the concepts of a knowledge domain are referred as "ontologies". In the literature, an ontology was defined as an “explicit specification of a conceptualization of a domain” [85]. Furthermore, an ontology represents a set of hierarchically structured terms, and provides multiple relations to bind objects together. This facilitates the extraction of meaningful inferences from the information created in a knowledge base.
These aforementioned functionalities are empowered by technologies such as RDF, OWL, SPARQL, etc., which we will next present in details.

2.4.1 RDF(S) and OWL

The Resource Description Framework (RDF) [53] is a World Wide Web Consortium (W3C) “model for data interchange on the Web”. RDF represents real world objects and relationships between them, by using URIs. The linking structure forms a graph, where the edges represent the named link between two resources, represented by the graph nodes. This graph-based representation is often called a “triple”, that is the association of a subject, predicate (i.e. property representing the relationship) and an object. RDF was extended later to be RDFS (S for Schema) that allows more expressiveness. It is thus possible to express subsumptions between entities, as well as setting restrictions on the relationships between them. However, there are some limitations when using RDF/RDFS, such as the inability to conduct automated reasoning on knowledge models. This is where OWL comes into the picture to fill this gap.

The Web Ontology Language OWL [133] is a family of knowledge representation languages based on Description Logic (DL) [13] with a representation in RDF. It forms an ontology by defining real world concepts, and their relationships in vocabularies. The concepts in an OWL ontology are named as classes, and relationships as

properties. OWL integrates, the same functionalities as RDF/RDFS, in addition to tools for comparing properties and classes: identity, equivalence, opposite, cardinality, symmetry, transitivity, disjunction, etc. Thus, OWL offers machines a greater capacity for interpreting web content than RDF/RDFS, due to its larger vocabulary and formal semantics. Moreover, OWL ontologies include axioms that assert constraints over their concepts and individuals. These axioms can be realized as simple assertions or as simple rules. It is also worth to mention that OWL make the open-world assumption, that is if a statement has not been defined explicitly, irrespective of whether it would be true or not, we cannot infer that the statement is false. *Table 2.2* summarizes the components of RDF(S) and OWL.

Moreover, the need of manipulating ontologies in dynamic environments, pushed computer scientists to implement the OWL API [92]. This latter is a Java interface and implementation for OWL, and contains a set of feature-rich interfaces, allowing the creation and management of ontologies. The API includes interfaces that define the bare bones of an OWL ontology, namely *OWLClass, OWLIndividual, OWLObjectProperty* and *OWLDatatypeProperty*.

2.4.2 SPARQL

As any knowledge representation, ontologies need to be queried to extract particular information. Thus, W3C proposed a standard query language for RDF, called SPARQL (Simple Protocol And RDF Query Language) [168].
SPARQL is "data oriented" in that it only queries information that are asserted in the knowledge model, and there is no inference in the query language itself. SPARQL does nothing more than take the description of what the application wants, in the form of a query, and returns that information, in the form of a set of links or an RDF graph. SPARQL also can be used to query an OWL model to filter out individuals with specific characteristics. Furthermore, the syntax of a SPARQL query is similar to SQL, and consists of triple patterns (RDF triples) where each of the subject, predicate and object may be a variable, conjunctions, disjunctions, and optional patterns. There are different types of a SPARQL query such as SELECT, ASK, CONSTRUCT, and DESCRIBE, that identify the variables to be included in the query response, along with a WHERE clause

Table 2.2: RDF(S) and OWL Components and Syntax

Semantic Web Standrad	Entities/Properties	Abstract Syntax
RDF	Instance	rdf:Description
	Instance relation	rdf:type
RDFS	Subclass relation	rdfs:subClassOf
	Domain of a property	rdfs:domain
	Range of a property	rdfs:range
	Subproperty	rdfs:subPropertyOf
OWL - Concepts	Top concept	owl:Thing
	Empty concept	owl:Nothing
	Ontology concept	owl:Class
	Class equivalence	owl:equivalentClass
	Class disjointness	owl:disjointWith
	Class intersection	owl:intersectionOf
	Class union	owl:unionOf
	Class negation	owl:complementOf
OWL - Properties	Data property	owl:DatatypeProperty
	Object property	owl:ObjectProperty
	Property equivalence	owl:equivalentProperty
	Inverse property	owl:inverseOf
OWL - Individuals	Instances equivalence	owl:sameIndividualAs
	Instances difference	owl:differentFrom
OWL - Restrictions	Restriction	owl:Restriction
	Restriction property	owl:onProperty
	Existential quantifier	owl:someValuesFrom
	Universal quantifier	owl:allValuesFrom
	Specific value	owl:hasValue
	Minimum cardinality	owl:minCardinality
	Maximum cardinality	owl:maxCardinality
	Cardinality	owl:cardinality

that defines the conditions that must be respected. The evaluation of the query is based on graph pattern (a set of triple patterns) matching. This graph pattern, located in the WHERE clause of the query, is defined recursively and contains triple patterns and SPARQL operators. *Figure 2.4* shows an example of a SPARQL query.

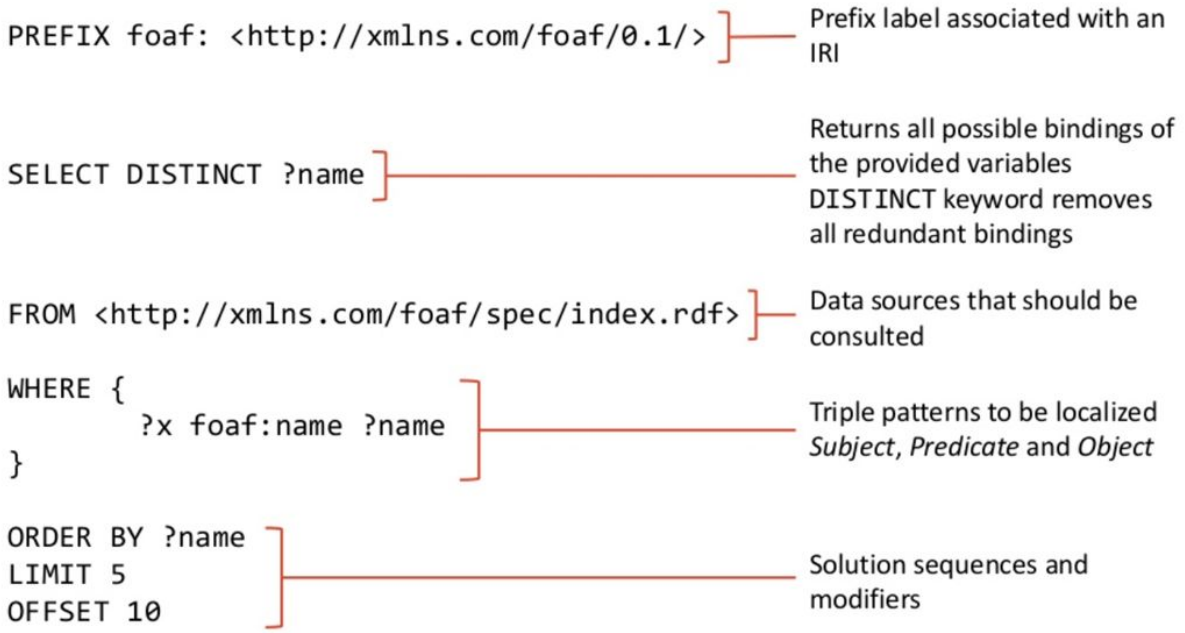


Figure 2.4: SPARQL query example

2.4.3 SWRL and SQWRL

The Semantic Web standards discussed so far permit certain types of rules to be defined but these are limited to classifications of objects. On their own, the Semantic Web languages RDF(S) and OWL do not permit definitions of Horn clauses, which limits the expressiveness of the rules they can define. Therefore, the Semantic Web Rule Language (SWRL)[93] was proposed for the Semantic Web, and is used to express rules as well as logic. Its syntax is of the form: *antecedent* $\rightarrow$ *consequent*, where both antecedent and consequent are conjunctions of atoms written $a_1 \wedge ... \wedge a_n$. The intended meaning can be read as: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold. Each atom can be formed from unary predicates (classes), binary predicates (properties), equalities or

inequalities, and variables are prefixed with a question mark (e.g., ?x). For example, the following rule: *hasParent(?x,?y)* $\wedge$ *hasFather(?y,?z)* $\rightarrow$ *hasGrandFather(?x,?z)*, asserts that the combination of the *hasParent* and *hasFather* properties implies the *hasGrandFather* property.

Moreover, SWRL was extended with some built-in libraries to facilitate some tasks, such as, directly creating new individuals in a rule.
For instance, the built-in *swrlx:makeOWLThing(?x,?y)* will cause an individual to be created and bound to *?x* for every value of variable *?y* matched in a rule.
We also distinguish a sub-language of SWRL, that is SQWRL (Semantic Query-Enhanced Web Rule Language), which provides SQL-like operators for extracting information from OWL ontologies (e.g., *owl:Thing(?i)* $\rightarrow$ *sqwrl:select(?i)*). SQWRL querying helps in achieving axioms that could not be expressed directly in SWRL because of the lack of existential quantification support in the language. SQWRL queries operate on known individuals of an OWL ontology and does not accumulate from within a rule, which means that query results cannot be written back to the ontology to not invalidate OWL's open world assumption and lead to non-monotonicity.

An implementation of SWRL is the SWRL API [151], that is a Java API for working with the OWL-based SWRL rule and SQWRL query languages. It includes graphical tools for editing and executing rules and queries.

2.5 Conclusion

This chapter discusses various concepts that are related to our research. First, we present the difference between the a priori and the a posteriori access control. Second, we summarize the most three common and expressive security policy models. Next, we discuss the a posteriori access control and its confusion with intrusion detection systems, and we review its existing literature.

In contrast, as the a posteriori access control is constituted of two essential elements, logs and the security policy, we study the classical log normalization methods and policy representations. We also highlight the advantages provided when modelling

both these elements with ontologies. Finally, we end this chapter by representing particular Semantic Web technologies which we will use alongside in this thesis.

Chapter 3

Extracting Log Information Using Semantic Mediation

3.1 Introduction

Log files are a huge asset in an organization as they contain vital information about the users and their actions in the system. The first step of the a posteriori access control is about assuring logging, namely, log processing. This step is fundamental as the analysis will be based on the extracted information. Therefore, managing logs have a lot to deliver to empower policy compliance evaluation with the true strength of log information.

Nevertheless, this process becomes more challenging with the increasing volume of logged data. Thus, digging in the vast amounts of information is not simple and requires analysts to acquire log formats. As previously discussed in *Section 2.2.1*, the multiplicity of log sources induces different log formats that may contain the same or different type of information. Moreover, the common techniques that are used to analyze logs such as normalizing logs into one format as well as filtering and finding patterns [90], are not as effective as they were before due to the growth of generated logs and the need to identify correlation between log events. Thus, these techniques require a significant time in processing, which ultimately converts into cost. In consequence, it is a matter to provide simple, efficient, and economical means to access data logs. Ideally, the solution must guarantee different criteria such as system autonomy, scalability, and transparency for accessing data location and format. In

contrast, in the context of the a posteriori access control, the extracted information should be relevant to the security policy to ease the analysis.

In this chapter, we provide a novel solution to extract information from logs using semantic mediation techniques. The goal is to resolve the heterogeneity between log formats in provenance of different log sources as a semantic mediator makes it possible to inter-operate various sources of information without modifying their internal functioning. Therefore, the log formats will remain intact and the information extraction will be done by querying logs; hence, we have a different view of "*log processing*", to be "*log query processing*", and by "*query processing*" we mean "*query rewriting*".

3.2 What is a Semantic Mediator?

The multiplication of data sources has made it impossible for a monolithic system to assimilate all the information. In contrast, the semantic mediation problem has been brought with the concept of a federated database, where several autonomous data sources wanted to coordinate with each other without being fully integrated so that distributed request plans are possible. Thus, to overcome this problem, [209] proposed an architectural model, where a software module is responsible for accessing a set of data sources while providing clients the illusion of using a single information system. This software module is called *mediator*, which becomes semantical when the data represents structured knowledge with formal semantics. As a result, a *semantic mediator* is based on models of knowledge representation that are able to describe, to a certain extent, the semantics conveyed by a piece of information and on tools to compare and unify the information semantics independently of the underlying structures. It can be responsible for locating data sources, to transmit queries to each source, or from one source to another, to retrieve the queries responses and possibly send them back to other sources [209]. Moreover, semantic mediators are essentially used for *Query Rewriting* [29], where queries are mediated from a single query access point to various data sources.

Yet, the notion of semantics of an entity cannot be represented in an absolute way.

It only makes sense when an entity is in relation to a particular context that can be represented by a concept map that describes a particular field of application. A concept is generally defined from the content of an ontology, that is a formal description of an abstract and simplified view of the world that one wants to represent.

That being said, a semantic mediator constitutes an intermediary mechanism, between different data sources, that uses ontologies to share a standardized vocabulary or protocol to communicate. Therefore, the support of query rewriting is done by exploiting the semantic relationships between the different sources schemas (ontologies), as semantic correspondences, namely "mappings", are defined by an administrator to express a query from one global source schema in terms of other target schemas.

3.3 Semantic Mediation For Access Control

Several researches were interested in using semantic mediation solutions in access control, namely for privacy-preserving enforcement. For instance, in [22], a Privacy-Preserving Service-Oriented Data Integration System (PAIRSE) was proposed. PAIRSE only allows access to information to which users are entitled to a given purpose. The queries in this project are resolved by automatically selecting and composing data services through the use of sophisticated query rewriting techniques to devise a novel service composition algorithm. Furthermore, [57] provided a solution to the problem of allowing interoperation while preserving the autonomy and security of the local sources by using wrappers and a mediator. The authors used query folding, to resolve the semantic heterogeneity of the information sources that was based on manually expressed rules. The work in [155] proposed a Semantic Access Control model (SAC) that extends RBAC, by considering the semantics of objects and associates permission with concepts instead of objects. Based on this model, a mediator-based interoperation system (SACE), was introduced to resolve semantic heterogeneity and enable access control in one process. It was also shown that SACE incurs only minor performance degradation in comparison to non-secure interoperation systems. Another effort for enabling privacy-preserving secure semantic access control was PACT [140]. PACT allows the sharing of data among heterogeneous databases while providing privacy

and confidentiality for metadata. It is a mediator-based solution, incorporating encrypted ontologies, encrypted ontology-mapping tables and conversion functions, encrypted role hierarchies, and encrypted queries. The encrypted query results are sent directly from the answering system to the requester, bypassing the mediator to further improve the security of the system. One of the distinctive features of PACT is that very few changes to the underlying databases are required.

Moreover, [201] showed how the specification and enforcement of authorization could be implemented in federated database systems. In addition, the authors in [170] introduced a concept-level semantic access-control for the Semantic Web, that deals with how access controlled resources names can be rewritten using other terms subject to logical rules expressed with OWL. Besides, an ontology-based rights expression language built on top of OWL to represent access rights of resources was presented in [171]. Finally, [4] proposed a Mediator Authorization-Security model to provide secure interoperation among heterogeneous semantic repositories. The authors addressed the issue of interoperability and trust incorporation into semantic interoperability. Despite the complexity of the mediator system, they showed how their model still provides acceptable performance.

All the above efforts used semantic mediation techniques to enforce the a priori access control. On the contrary, our goal is not to preserve access to data, but rather to extract useful information for the a posteriori access control. We will thus take advantage of the benefits that offers the semantic mediator to query different log sources with multiple formats, and have results in terms of the security policy.

3.4 Semantic Mediation in the a Posteriori Access Control

In an a posteriori access control system, policies are checked after granting access to users. Once authenticated, access to information will be governed by an access control policy that is contextual to the application domain. A reconciliation between policy rules and logged actions is then needed, in order to verify whether access rules are fully respected or not. Therefore, we define a particular setting in which we deploy a semantic mediator to extract information from logs in terms of the security policy, by

considering the following:

- There are multiple log sources denoted as $S_1, S_2, ..., S_n$ and each log source has a particular format denoted as $f_1, f_2, ..., f_n$.
- The security policy is denoted as P and is represented in an ontological model according to ABAC. Details about this policy are provided in *Section 4.3*.
- A semantic mediator exists between the policy and the logs for query processing.
- The provided logs are well structured to enable retrieving more meaningful information from them [104].

Now that we defined our information extraction setting, we consider that the queries are sent automatically from the defined security policy to the logs. The semantic mediator will then proceed in rewriting the query expressed on one source schema into another request expressed on a target schema. This rewriting process is done using previously established semantic correspondences between the different schemas (ontologies in our case). In addition, to have a unified final result we divide the rewriting process into two stages: *Semantic Query Rewriting* and *Syntactic Query Rewriting*. In the following, we present the needed setup of the mediator to perform both query rewriting types, and discuss each stage.

3.4.1 Semantic Mediator Setup

The use of the mediation approach allows information to be retrieved dynamically from original log sources at query time. We adopt a conceptual model rather than a logical one to manage log querying easily. Thus, our semantic mediator is ontology-based, where each log source is represented with an ontology. Moreover, a consensual ontology is needed to represent the application domain, in addition to mappings between this latter and the ontologies representing log sources. These mappings are used to rewrite the global query into a union of queries that match local ontologies. The major advantage of extracting information from logs using this approach is that it allows different log sources to be integrated, while enriching their querying with ontological knowledge.

Ontologies for a conceptual view of logs

Local Ontologies. Each log source contains a huge number of raws where each raw represent a log event. In general, a log event appears in a specific format that is proper to the log source, and that contains a limited number of fields, which values vary from one log event to another. Moreover, the fields' types are normally known when configuring a log source; hence, can be represented as concepts in an ontology and can have relationships with one another. Therefore, local ontologies are created to provide a conceptual view of log sources. These ontologies can be designed by experts to represent the field names managed by each source or can be semi-automatically generated using suitable tools (e.g., RDBtoOnto[39], XS2OWL[194], etc.). In our case, the ontologies are statically perceived.

That being said, in the mediator, each log source S_i will be viewed through an ontology O_i that contains the demonstrated fields in its format f_i. It is worth to mention that none of the S_is will be modified. Local ontologies serve as a conceptual access point to the log source's data that is used during the interactions between the mediator and the log sources. Thus, logs will not be transformed into ontologies, and local ontologies will only contain the main concepts provided by each log source, rather than the values appearing in each event (individuals). For example, considering a database log that contains the following columns: *UserID*, *Action*, and *TimeLogged*; only these concepts will appear in the ontology and not their values e.g *"100"*, *"View"*, and *"2019-02-11 21:31:48"*, respectively. We should always remember that the logs will remain intact, and that the ontologies are used for query rewriting purposes.

Global Ontology. The global ontology (also referred as domain ontology), constitutes the entry point from which the queries sent to the logs are mediated. Thus, it should provide a global consensual conceptual level of the application field and a structured vocabulary for querying the relevant log sources. Until the day, there is no standard format that can represent all log types including application logs, as these latter are usually determined by the developer of the software program. Nevertheless, even if log contents may vary a lot from one source to another, they all have a common thing: all of them simply register the event that occurred, more precisely, "what happened?

when? by whom?". Therefore, the concepts that form the global ontology are every log type essential elements, that are the *Subject*, *Action*, *Object*, and *Timestamp*, and which are defined relatively to a log event.
It must also be noted that this design of the global ontology will permit querying the logs in terms of the security policy since access control models are usually based on the three entities *Subject*, *Action*, and *Object*, and when they are expressive enough they consider the *Time* as a contextual condition. Moreover, the concepts *Subject* and *Object* can refer to the representatives of subjects and objects that are defined in the logs such as subject and object attributes. This functionality is assured using mappings, which we discuss further on. This domain ontology is presented in *Figure 3.1*, and is denoted as O_G.

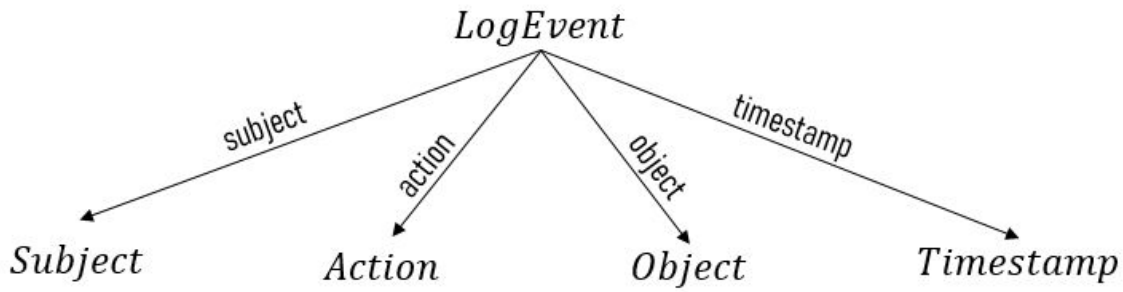


Figure 3.1: Global Log Ontology O_G.

Mappings between Ontologies

The goal of ontology mappings is to allow the retrieval of information from log sources through query rewriting. Using ontology mappings, a query expressed in terms of the global ontology can be rewritten into a union of queries that are expressed over each local ontology representing a log source.

Furthermore, since the adopted schemas in the mediator are ontologies, the initial input queries are expressed in SPARQL, and over O_G. It is evident that to have a successful query rewriting process, mappings between O_G and each O_i should be established. We thus define a mapping as follows.

Definition 3.4.1. ***(Mapping)***
A mapping is a set of correspondences between different entities of different ontologies.

Definition 3.4.2. ***(Correspondence)***
Let O_1 and O_2 be two ontologies. A correspondence μ is a triplet <e_1, e_2, r> where

- *e_1 and e_2 are two alignable entities of O_1 and O_2 respectively.*
- *$r \in R$ denotes an existing relation between e_1 and e_2.*

An entity in an ontology can be a class, an object property, a datatype property, or an individual. In our case, individuals do not exist in the ontologies, so there will not be any relative entities. The relationship between entities can be an equivalence ($\equiv$) or a subsumption ($\subseteq$). Additionally, complex expressions in the correspondences between entities can be found as well, using union ($\cup$) and intersection ($\cap$) operations. For example, μ: O_G:Timestamp $\equiv$ O_1:Date $\cup$ O_1:Time.

Moreover, different strategies can be adopted for defining semantic correspondences between the global and local ontologies, from which we cite *Global-As-View (GAV)* and *Local-As-View (LAV)*. In the *GAV* approach, each entity in the global ontology is defined as a view of the different log sources ontologies to be integrated. A major advantage of this approach is that answering a query is quite trivial with reference to the overall schema. This means that the received requests can be easily rewritten with the terms used by each local source. In contrast to the *GAV* approach, in the *LAV* approach the views on the sources define how local information is related to the global schema by expressing a correspondence between each relationship in the local schema and one or more relationship in the global schema. The main advantage of the *LAV* approach compared to *GAV* is that there is no dependency on the overall pattern. Therefore, the addition of new sources to the system only requires the definition of the necessary mappings between the source schema and the overall pattern. However, in this approach, responding to a query becomes more difficult because rewriting a query is difficult to do.

In our case, the global ontology is not subject to change and the information to be retrieved from the local log sources are in fact dependent of it. Moreover, in reality,

new log sources are not added frequently in an application domain. Therefore, we follow the *GAV* approach do define the mappings or the semantic correspondences between the global ontology and each of the local ontologies. It must also be pointed out that since our choice of global ontology is static and is not incrementally built as new sources join the mediator, adding a new source to the mediator will not be complicated as in the usual *GAV*, and will only require the definition of new mappings as in the *LAV*.

Besides, mappings can be done manually or semi-automatically to set correspondences between each of the concepts *Subject*, *Action*, *Object*, *Timestamp* in O_G, and their relative concepts in each O_i. It is worth noting that although many efforts have been made to automate the generation of mappings or alignments between ontologies (e.g., Align API [56], COMA++ [9], etc.), we consider that the mappings are at best formed semi-automatically as an expert is always needed to check the generated results (that are normally score-based).

3.4.2 Query Rewriting Process

At this point, ontologies are defined and their corresponding mappings are established. Thus, the setting allowing the query rewriting process is ready. In the a posteriori access control, this process is governed by the security policy P. Its main goal is to retrieve information, from different log sources, that is relevant to the security policy to permit compliance checking. The first step consists in sending a SPARQL query in terms of the global log ontology O_G. Next, the mediator starts the rewriting process to generate other queries that are understandable by the log sources and thus, have a response to the initial query. Inside the mediator, the query is undergone two transformations that are the *Semantic Rewriting* and the *Syntactic Rewriting*, and which are performed subsequently. The resulting queries will be executed on the concerned log sources to extract information. These log sources are identified by resolving the mappings that a requested attribute in Q_G has. We thus detail each rewriting step in the following.

Semantic Query Rewriting

The semantic query rewriting is about conserving the language of the query, while expressing it over another source ontology. This can be done by resolving the mappings that exist between two ontologies. For instance, a SPARQL query will remain a SPARQL query and the only modification will consist in replacing its entities with their semantic equivalents. Therefore, the semantic mediator takes a query Q_G expressed over O_G as input, decomposes it into multiple subqueries if needed, and rewrites it (or its subqueries) to a semantically corresponding SPARQL query Q_i. The generated Q_i is expressed in terms of the concerned O_i with respect to the mapping M_i that exists between O_G and O_i.

We define SP as the domain of SPARQL queries, M as the domain of mappings between O_G and O_i, and $SemRW$ as the function responsible of the semantic rewriting of a SPARQL query:

$$\begin{aligned} SemRW : SP \times M &\rightarrow SP \\ (Q_G, M_i) &\rightarrow SemRW(Q_G, M_i) = Q_i \end{aligned} \tag{3.1}$$

The rewritten query is generated by replacing the graph pattern of the initial query with the rewritten graph pattern. Variables appearing in the rewritten graph pattern are the same as the variables that appeared in the initial graph pattern. In addition, the rewriting process is independent of the query type (i.e., Select, Ask, etc.), the SPARQL solution sequence modifiers (i.e., Order By, Distinct, etc.), and the SPARQL algebra operators (i.e., Union, Optional, etc.). Since a lot of works treated the SPARQL rewriting problem, we refer to [130] for more rewriting rules details.

Syntactic Query Rewriting

In this second step, a syntactic transformation of the rewritten SPARQL queries (each Q_i) will be achieved. The syntactic rewriting consists in changing the syntax of the query (the language it is expressed in), while maintaining its semantics.

On the other hand, different concepts can be used to structure the information in log files such as relationship in the relational model, XML tag, CSV, etc. Thus, the

SPARQL query can be converted to an SQL query, XQuery, or any other type of query depending on the existing log formats. The use of structured logs is advantageous because it favors automation. Extracting useful content from unstructured logs requires accounting for their structure because data semantics highly depends on relations between neighboring data elements. Thus, structured logs ensure that all relevant data along with relationships between them are captured from the correct regions in log files, and guarantee correct mappings, between the log source and its conceptual ontology, if they exist. Besides, they provide both completeness and contextual correctness.

Let QR be the domain of all query types excluding SPARQL. We define the function $SynRW$ for syntactically rewriting a SPARQL query as follows:

$$\begin{aligned} SynRW : SP \times f &\rightarrow QR \\ (Q_i, f_i) &\rightarrow SynRW(Q_i, f_i) = q_i \end{aligned} \tag{3.2}$$

knowing that q_i is executable on f_i.

For each log storage format, specific algorithms for syntactically rewriting SPARQL should be defined. Moreover, mappings m_i between log sources and their corresponding local ontologies can also exist depending on the rewriting algorithm, and the source's type format. These mappings can also be specified manually or automatically.
Finally, q_i will be executed on S_i, and all the obtained answers will be combined to respond to the initial Q_G. The proposed solution is presented in *Figure 3.2*. The goal of this chapter is not to develop new query rewriting algorithms, but rather to provide a novel solution to extract information from multiple log sources, in the case of the a posteriori access control. In consequence, and without loss of generality, we will further treat the case of two log formats, that are commonly used in organizations, that are logs in the relational model and in XML, since the corresponding syntactic rewriting algorithms of SPARQL already exist in the literature [28, 64].

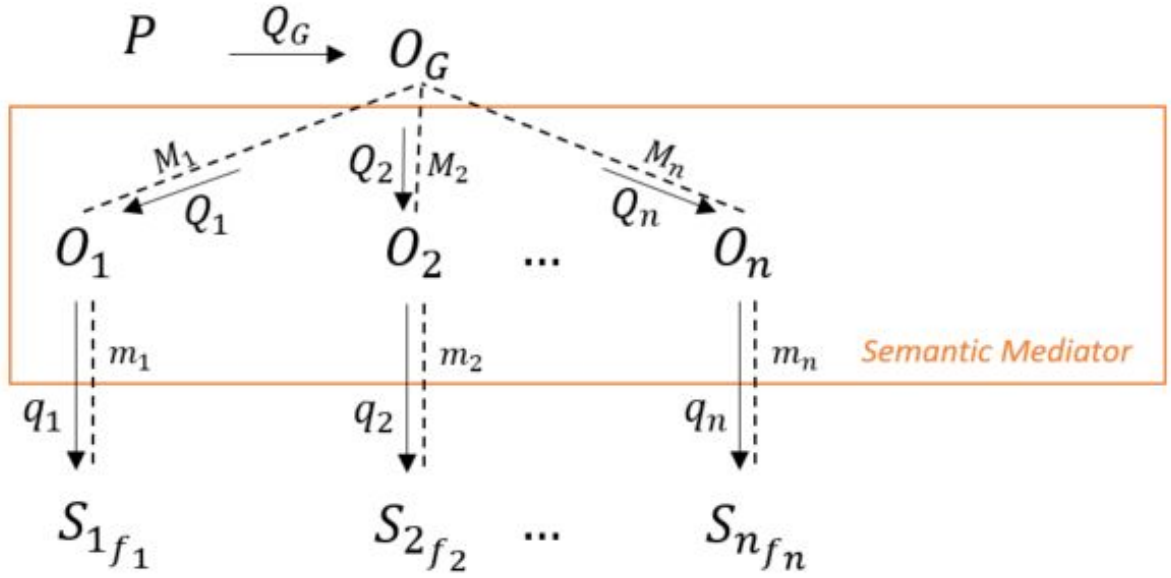


Figure 3.2: Query Rewriting Process.

3.4.3 Policy Reconciliation

From the obtained query results, corresponding axioms and assertions will be generated. Given that on an abstract level, the expression of any policy includes a set of quadruples <*subject, action, object, time*>, it is possible to establish links between the query responses and the security attributes used to express the access control policy, to check their compliance and detect if there was any violation. In contrast, when the deployed security policy is expressive, the extracted information from logs might not be enough to check policy conformity, as additional data is needed. This problem of enriching log information semantically is treated in *Chapter 4*.

3.5 Example Scenarios

In this section, we develop some example scenarios, inspired from real use cases, to demonstrate the practicality of our approach. The scenarios present situations that can occur in the healthcare domain, particularly when using an Electronic Health Record (EHR) system. An EHR system is a new way to store and process health information that supports continuity care, education, and research, and covers the need of all

engaged parties including patients, doctors, healthcare providers, and policy makers. Thus, EHR systems present a formidable "trustworthiness" challenge, which makes it a suitable environment to deploy the a posteriori access control.

3.5.1 Scenarios

Two hospitals A and B use an Electronic Health Record (EHR) application to share information between each other. However, the server in hospital B generates logs in a database table, while hospital A's server generates XML logs. The two servers record almost the same information about the users' actions in the application domain. Evidently, the users appearing in the logs of each server correspond to the employees of the corresponding hospital.

Scenario 1

In January 2019, a patient X entered the emergency room in hospital A. In order, to access to his medical record, hospital A asks hospital B to send her the patient's medical history. The patient's designated healthcare professional (HCP) from hospital B sends the patient's medical record to hospital A. Two weeks later, this same patient went to consult his designated HCP in hospital B, when his HCP noticed that there was something wrong in the prescription given from hospital A.
This fact triggered the investigation process to search for the principal cause of the prescription mistake.

Scenario 2

A certain HCP in hospital B took a 4-day leave from work for illness. In consequence, a substitute HCP was called to replace him during this period. On his return, the HCP would like to know which medical records have been modified during his absence, for patients follow-up reasons.

Scenario 3

Going deeper in scenario 1, the reason why the patient went to consult his HCP in hospital B, was his affection with a very low blood pressure, in addition to a lot of vomiting. The error in the prescription was that the medicine prescribed from hospital A is not compatible with the patient's previously prescribed medicine, when he had a bacterial pneumonia, a less than one month before.

3.5.2 Synthetic Logs Generation

One open-source EHR aplication is iTrust [136]. Therefore, to generate transaction logs, we deployed it on an Apache Tomcat server, and we simulated users' actions using JUnit test scripts. Moreover, we configured it in a way to produce Database (MySQL) and XML logs for a particular group of users (to differentiate between the employees of each hospital).

Figures 3.4 and *3.5* show excerpts of the generated logs on each hospital's server, supposedly configured by their respective security administrators. Besides, the corresponding generated ontologies and mappings are shown in *Figure 3.3*.

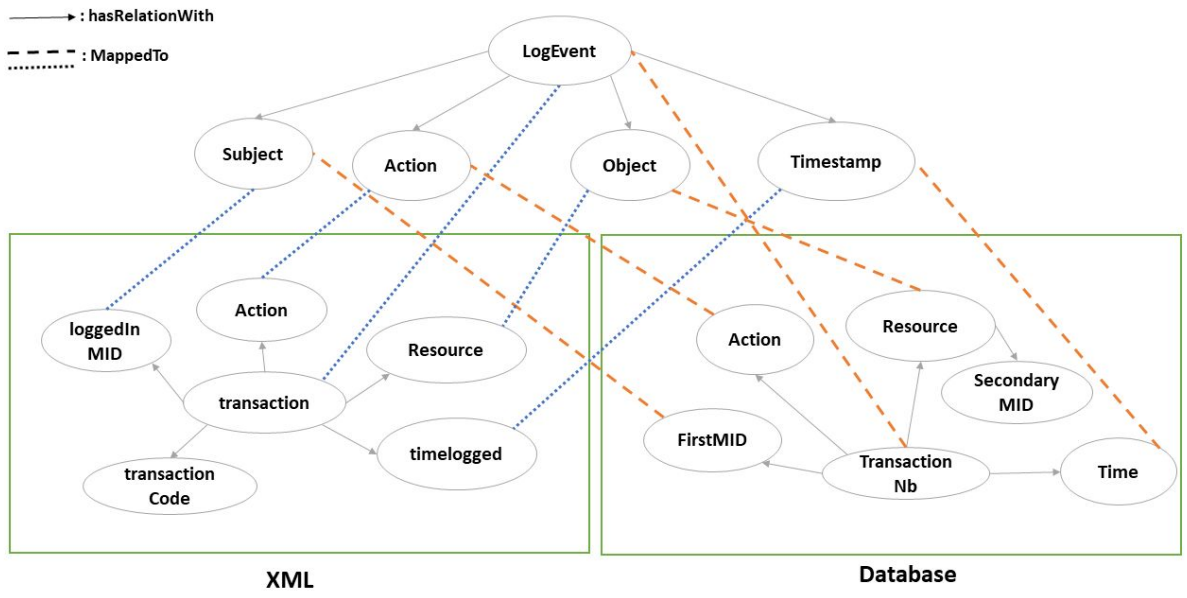


Figure 3.3: Mappings between ontologies.

```
<transaction id="1378" xmlns:action="http://hospital.com/EHR">
<timelogged>2019-01-07 16:42:30</timelogged>
<loggedInMID>9000000003</loggedInMID>
<Action>ADD</Action>
<transactionCode>2210</transactionCode>
<Resource>APT314450</Resource>
</transaction>

<transaction id="3623" xmlns:action="http://hospital.com/EHR">
<timelogged>2019-01-09 10:03:51</timelogged>
<loggedInMID>9000000003</loggedInMID>
<Action>VIEW</Action>
<transactionCode>900</transactionCode>
<Resource>MR314160</Resource>
</transaction>

<transaction id="3674" xmlns:action="http://hospital.com/EHR">
<timelogged>2019-01-09 10:37:04</timelogged>
<loggedInMID>9000000003</loggedInMID>
<Action>ADD</Action>
<transactionCode>1120</transactionCode>
<Resource>LP314160</Resource>
</transaction>

<transaction id="4229" xmlns:action="http://hospital.com/EHR">
<timelogged>2019-01-10 12:24:38</timelogged>
<loggedInMID>5000000001</loggedInMID>
<Action>VIEW</Action>
<transactionCode>900</transactionCode>
<Resource>MR314160</Resource>
</transaction>
```

Figure 3.4: XML Log.

TransactionNb	FirstMID	Resource	SecondaryMID	Action	Time
265	8000000011	MR314980	314980	VIEW	2019-01-08 18:32:59
544	9000000013	MR314160	314160	VIEW	2019-01-09 10:15:01
545	9000000013	MR314160	hospA	SEND	2019-01-09 10:15:13
1002	9000000085	MR322660	322660	EDIT	2019-01-10 09:48:27

Figure 3.5: Database Log.

```
<align:map>
  <align:Cell>
    <align:entity1><Class rdf:about="GlobalOntologyIRI#Object"/></align:entity1>
    <align:entity2><Class rdf:about="XMLOntologyIRI#Resource"/></align:entity2>
    <align:relation>=</align:relation>
    <align:measure rdf:datatype="&xsd;float">1.0</align:measure>
  </align:Cell>
</align:map>
```

Figure 3.6: Example of mapping in EDOAL.

3.5.3 Mediator Implementation

The objective of using a semantic mediator is to enforce the information extraction from logs, in the posteriroi access control. Therefore, we built our semantic mediator by combining different existing open source tools.

To accomplish the semantic rewriting of a SPARQL Query (SPARQL - to - SPARQL), we used a publicly available toolkit for ontological mediation over RDF [135]. This tool rewrites the initial SPARQL query, taking into account the mapping representation, between the global ontology and the different local ontologies, expressed with the Expressive and Declarative Ontology Alignment Language (EDOAL) [180]. EDOAL is a highly expressive and serializable language built upon the Alignment Format [56], a well-known specification extensively used for representing alignments in ontology matching tasks. *Figure 3.6* shows how mappings are expressed in EDOAL.

However, this toolkit has some limitations since it supports only SELECT and CONSTRUCT queries, and is not able to rewrite the SPARQL query when there is a complex correspondence between the different ontologies' entities using the union operator. We can overcome this limitation by extending the tool with a function that handles this case. For the sake of simplicity, our defined mappings are currently limited to the exact equivalence of two different entities from two different ontologies.

As for the syntactic query rewriting (SPARQL - to - OtherTypeOfQuery), we were interested in converting SPARQL to both SQL and XQuery for test purposes. Many efforts have been made in the literature to perform this task, from which we cite [28, 64]. Nevertheless, we relied on open source tools.

For rewriting SPARQL into SQL we used Ontop [33]. Ontop is an open-source

Ontology-Based Data Access (OBDA) system that maps data sources to ontologies representing the domain of interest, and through which querying these relational data sources is possible. Advantages of Ontop are its compliance to all relevant W3C recommendations (including SPARQL queries, R2RML mappings, and RDFS ontologies), and its support for all major relational databases. Furthermore, each mapping axiom defined in Ontop corresponds to a pair of source and target. The source is an SQL query over the database, and the target is a graph pattern that contains placeholders that refer to the column names mentioned in the source query. These mapping axioms generate RDF triples, by replacing the placeholders in the target with the values returned when evaluating the source SQL query.

As for converting SPARQL to XQuery we used the open-source SPARQLToXQuery [191]. This tool handles only SPARQL SELECT queries in three different cases: (1) the subject and object are variables, (2) the subject is a variable and the object is a literal, and (3) the subject is a variable and the object is an URI. The fact that it only allows the subject of a triple pattern to be a variable, makes the Object and Datatype properties correspond to a subchild of an element in the XML file. Thus, the domain of the property will refer to the parent element, and the range will correspond to its subchild value. It is also worth to mention that the SPARQLToXQuery tool is made to address RDF/XML data. We modified it so that it queries XML.

Figure 3.7 shows our open-source based semantic mediator architecture.

3.5.4 Query Rewriting Applied in the Scenarios

Starting with *Scenario 1*, and considering that the patient's Medical ID (MID) is 314160, the investigation consists of searching for the actions done, by which subjects, in January 2019, on this patient's medical record. The medical record of this patient is identified by "MR314160".
The query rewriting for this investigation is shown in *Table 3.1*.

The initial SPARQL query is transformed into a conjunction of SPARQL queries expressed in terms of the local ontologies. For instance, the object properties *action*, *subject*, *timestamp*, and *object* from the global ontology are mapped to the object properties *Action*, *loggedInMID*, *timelogged*, and *Resource*, and *action*, *executedBy*,

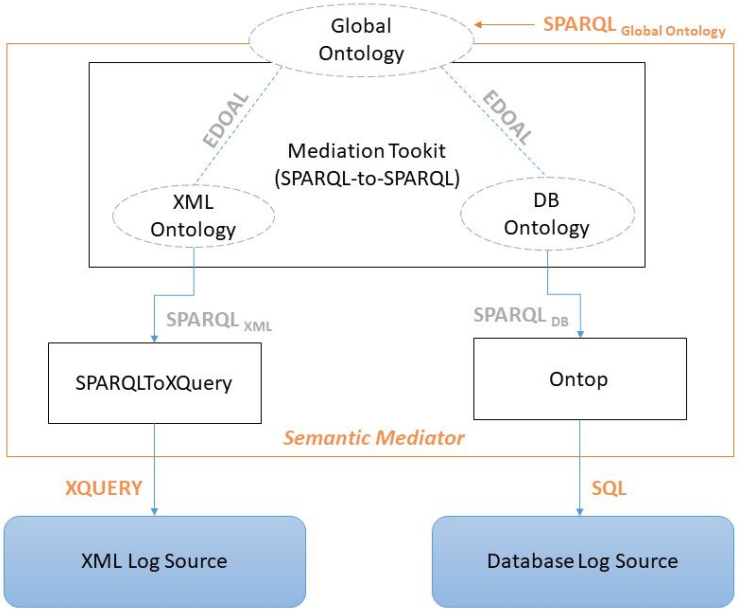


Figure 3.7: Semantic Mediator Architecture.

executedAt, and *executedOn* from the ontologies representing the XML log and the Database log, respectively. Afterwards, each of the resulted SPARQL queries will be syntactically transformed depending on the underlying log structure. From SPARQL to XQuery, the *transaction* class of the XML ontology refers to the *transaction* element of the XML log and the object property *loggedInMID* refers to the subchild *loggedInMID* of the element *transaction*. Besides, the other SPARQL query is converted to an SQL query, based on the mappings defined in Ontop. Excerpts of these mappings are shown in *Figure 3.8*.

As for *Scenario 2*, we suppose that the Medical ID (MID) of the substituting HCP is "9000000085". Thus, the query is about retrieving the resources that this HCP has edited. Using the same mappings as *Scenario 1*, the SPARQL query is subsequently rewritten semantically and syntactically. Since both HCPs executed their actions in Hospital B, it is obvious to not get an answer from the source log of Hospital A. The query rewriting process of this scenario is shown in *Table 3.2*.

We note that log, db, xml shown in the tables refer to the prefix URI of each ontology.

Moving on to *Scenario 3*, we consider that the logs have a finer granularity where the medicines prescribed are logged too, and that more complex mappings are defined

```
mappingId   Resource
target      :{Resource} a :Resource  .
source      select "Resource" from "table_log"

mappingId   M1
target      :{TransactionNb} :executedAt :{Time}.
source      select "TransactionNb", "Time" from "table_log"


mappingId   M2
target      :{TransactionNb} :action :{Action}.
source      select "TransactionNb", "Action" from "table_log"
```

Figure 3.8: Mappings defined in Ontop.

between the ontologies (e.g., the class Object in O_G is mapped to more than one class in O_i). The query consists then of searching for the doctors who prescribed the conflicting medicines, medicine 1 (med1) and medicine 2 (med2), for this patient, on a 2 month period. We also consider that a query decomposition layer is added to the mediator, which will be used before performing any rewriting. Therefore, the corresponding query of this investigation will be:

```
SELECT ?x ?y ?z WHERE {
   ?t log:action ?x.
?t log:object log:MR314160.
 {?t log:object log:med1.}
          UNION
 {?t log:object log:med2.}
   ?t log:timestamp ?z.
FILTER regex(?z, "^(2018-12|2019-01)")}
```

And will be decomposed into two queries, each one relating to one medicine: $SPARQL_1 \cup SPARQL_2$ where $SPARQL_k$=

```
SELECT ?x ?y ?z WHERE {
   ?t log:action ?x.
?t log:object log:MR314160.
```

```
?t log:object log:med_k.
?t log:timestamp ?z.
FILTER regex(?z, "^(2018-12|2019-01)")}
```

These resulting subqueries will be rewritten according to the different defined mappings and will be sent to each log source, that has the concept medicine. The rewriting process of each subquery is similar to the one shown in *Table 3.1*.

The obtained answers can form quadruples <*subject*, *action*, *object*, *time*>, to compare them with the rules defined in the security policy, and detect possible violations. However, if the security policy is modelled with a higher level of expressivity, for example, according to ABAC or OrBAC, we will need to enrich these results with more attributes. For instance, in *Scenario 1*, the LDAP directory can be consulted to check the roles associated with the extracted MIDs. Therefore, a possible violation can be that the medical record of the patient was consulted and edited by a Lab Technician, who is not supposed to be allowed to do that. As for *Scenario 2*, we can fetch in a database to see if the modified medical records are not related to other than the patients who had an appointment during that period of time. This problematic is treated in *Chapter 4*. Moreover, taking decisions about the accountability of the user when violations are detected is discussed in *Chapter 6*.

3.6 Discussion

Every a posteriori access control is built on the base of log processing, more precisely, extracting information from logged data. It is a very important step, since it is the starting point from which the analysis begins, to lead to decisions and set responsibilities. Thus, the use of semantic mediation techniques to accomplish this mission offers many advantages that we detail below.

To start with, it is *economical in terms of processing*. Unlike the existing log management tools, our approach neither parses nor filters provenance logs. The only process it has is the Query Rewriting process, which is quite fast since only one query is handled at a time. The duration of query rewriting and execution is in the range of 300 ms, which is evidently less than any parsing time that varies relatively to the log file size.

Next, it provides *scalability*. Our model is scalable since each data source is autonomous and independent from the other sources. New data sources can be added to the model. As the use cases showed how the approach can work for both XML and Database logs, other log formats could be considered. For instance, for a CSV file, we will need to implement a SPARQL to R rewriting algorithm to fulfill the need. However, this current architecture can support CSV files since they can be queried with SQL using specific (Java) libraries. One limitation can be that this approach is only suitable for structured or semi-structured log files, since ontologies and mappings have to be defined in advance.

Moreover, the use of SPARQL as a query language enables us to reap the benefits of *federation*, thereby it makes all the log sources look like one big database. Representing the different log formats in RDF serves as a standard *lingua franca* (least common denominator). As such, querying RDF with SPARQL hides the details of a source's particular data structure. This *reduces costs* and *increases robustness* of our model that issues queries. Furthermore, SPARQL enables specific questions to be sent to the logs to *retrieve* directly the *precised information* instead of sending queries with limited number of operations to get an answer.

Besides, the use of the semantic mediation solves the problem of the disparity of the multiple log sources, and makes them interoperable.

Last but not least, our proposal satisfies the requirements of the environment in which the a posteriori access control is deployed, such as the *end-to-end policy enforcement*. It is an end-to-end like question/answer system, from the security policy to the logs. All the query treatments are done transparently in the semantic mediator.

3.7 Conclusion

In this chapter, we proposed a new solution for an a posteriori log analysis based on a semantic mediator. We pictured how it can resolve the heterogeneity between log sources and enforce the information extraction. Moreover, we showed how all log events could be assimilated to a tuple <*subject, action, object, timestamp*>, regardless of their configured format. This nomenclature is well adapted for policy

compliance evaluation as it contains the basic concepts of any security policy. Besides, we decomposed the query rewriting process into two stages that are *Semantic Rewriting* and *Syntactic Rewriting*, and discussed how they can be done using ontology mappings. To prove our approach, we employed existing open source tools to build our semantic mediator, and presented its functioning in different scenarios in the healthcare domain. Despite the limitations that they imposed, we showed how our idea can be efficient and economical by testing it on both Database and XML logs.

Now that we treated the first step of the a posteriori access control, the next step is about analyzing the extracted information. However, when having an expressive security policy such as ABAC or OrBAC, additional information should be fetched to add more semantics and context to the extracted log event. Therefore, in the next chapter, we will treat the problem of the semantic enrichment of logs as well as policy temporal compliance.

Table 3.1: SPARQL Rewriting Process in Scenario 1

Original SPARQL Query	
SELECT ?x ?y ?z WHERE {?t log:action ?x; log:subject ?y; log:timestamp ?z. Filter regex(?z, "^2019-01") ?t log:object log:MR314160. }	
Rewritten SPARQL with XML Mappings	**Rewritten SPARQL with DB Mappings**
SELECT ?x ?y ?z WHERE { ?t xml:Action ?x ; xml:loggedInMID ?y ; xml:timelogged ?z ; xml:Resource xml:MR314160 . FILTER regex(?z, "^2019-01") | }	SELECT ?x ?y ?z WHERE { ?t db:action ?x; db:executedBy ?y ; db:executedAt ?z; db:executedOn db:MR314160 . FILTER regex(?z, "^2019-01") }
Generated XQuery	**Generated SQL Query**
import module namespace rdffunc; let $ts := doc('log.xml')//* for $t in $ts let $xs:=$t/Action for $x in $xs let $ys:=$t/loggedInMID for $y in $ys let $zs:=$t/timelogged for $z in $zs where $t/Resource='MR314160' and matches($z,"^2019-01") return <result> {rdffunc:objectResult($x,$xs)} {rdffunc:objectResult($y,$ys)} {rdffunc:objectResult($z,$zs)} </result>	Select Action, FirstMID, Time FROM table_log WHERE Time REGEXP '^2019-01' AND Resource= 'MR314160';
Query Response	**Query Response**
<result> <literal>VIEW</literal> <literal>9000000003</literal> <literal>2019-01-09 10:03:51</literal> </result> <result> <literal>VIEW</literal> <literal>5000000001</literal> <literal>2019-01-10 12:24:38</literal> </result>	db:VIEW, db:9000000013, db:2019-01-09 10:15:01, db:SEND, db:9000000013, db:2019-01-09 10:15:13,

Table 3.2: SPARQL Rewriting Process in Scenario 2

Original SPARQL Query	
SELECT ?x WHERE {?t log:action log:EDIT; log:subject log:9000000085; log:object ?x. }	
Rewritten SPARQL with XML Mappings	**Rewritten SPARQL with DB Mappings**
SELECT ?x WHERE { ?t xml:Action xml:EDIT ; xml:loggedInMID xml:9000000085; xml:Resource ?x . }	SELECT ?x WHERE { ?t db:action db:EDIT; db:executedBy db:9000000085; db:executedOn ?x . }
Generated XQuery	**Generated SQL Query**
import module namespace rdffunc; let $ts := doc('log.xml')//* for $t in $ts let $xs:=$t/Resource for $x in $xs where $t/Action='EDIT' and $t/loggedInMID='9000000085' return <result> {rdffunc:objectResult($x,$xs)} </result>	Select Resource FROM table_log WHERE Action='EDIT' AND FirstMID='9000000085';
Query Response	**Query Response**
NO ANSWER	db:MR322660,

Chapter 4

A Posteriori Violation Detection with a Static Policy

4.1 Introduction

As stated in the previous chapters, logs are the central part of auditing in the a posteriori access control, that is reviewed for action legitimacy checking and accountability purposes. Therefore, it is important to have meaningful, yet relevant logged information that permits to compare what happened with what is supposed to happen (security rules).

After having extracted information from logs using a semantic mediator, as presented in *Chapter 3*, the a posteriori access control moves to its second stage that consists in analyzing this information to detect violations of the security policy. Thus, to have an effective violation detection mechanism, log information should provide meaningful evidence. [31] addressed the question of which information should be included in logs for meaningful a posteriori compliance control. However this is rarely respected, and useful information can be found somewhere else than logs. In consequence, a valid explanation of policy conformity should exist, and the validity of this explanation relies on the availability of the necessary information for assessing policy compliance. On one side, expressive security policy models such as RBAC [70], ABAC [94], OrBAC [62], etc., assign permissions indirectly to the users through their attributes, and sometimes object attributes and contextual constraints as well. On the other side, logs do not trace this kind of information in general. For instance, in a

security policy defined according to the RBAC model, access rights are assigned to roles instead of individual users. Meanwhile, users' roles cannot be found in logs, where they are presented by their usernames or IP addresses instead. In consequence, establishing links between the explicitly defined attributes in the security policy, and the logged data is not that evident. This leads to the need to semantically enrich logged data with complementary information, in such a way, log analysis is accurate enough to make fair decisions when violations are committed.

Conversely, when performing an a priori access control, access attributes values are checked at the time of the access request. As a consequence, the system guarantees the respect of the security rules when granting access to the user. However, in the a posteriori access control, a lot of changes in the security attributes can take place between the time of access and the time of investigation (change of role, role delegation, change of status, etc.), and contextual conditions evolve between accesses (e.g., emergencies) [51]. Therefore, it is important to verify that the access attributes' values and conditions were the same as those defined in the security policy *at the time* when the information resource was accessed. This is similar to the case of forensics for criminal investigations, where the importance does not reside in where the suspect is now, but in where he/she was when the crime was committed.

This chapter has two goals that we accomplish as a bundle. The first one is to semantically enrich the extracted information from logs with complementary data for a more accurate comparison with the security policy. Thus, we propose a multi-agent system to perform this laborious information gathering task. The second goal is to ensure the temporal compliance of the collected attributes with the security policy, in other words, check if the attributes of the event had the right values at access time. This is achieved through the use of the Event Calculus (EC), a formal language for representing and reasoning about dynamic systems, which we express in SWRL, and that we integrate in the multi-agent system.

4.2 Materials

4.2.1 Multi-Agent System Definition

Complex and heterogeneous fields such as decision support in subtle situations, pattern recognition, and industrial process control have revealed the limits of the classical approach of Artificial Intelligence (AI) that is based on centralizing the expertise within a single expert system. Therefore, research in the AI field led to the birth of a new discipline that is the *Distributed Artificial Intelligence (DAI)*, also called *Multi-Agent Systems (MAS)*. Such systems are composed of distributed computation units, that engage in flexible, high-level interaction with one another and with their environment as well [47], that are called agents.

Definition 4.2.1. ***(Agent)***
An agent is a computer system, located in an environment, that acts autonomously to achieve the objectives (goals) for which it was designed. [210]

One can speak of autonomy because the agent's behaviour depends at least partially on its experience. It may act without the direct intervention of a third party (e.g., a human) and control its actions as well as its internal state. Moreover, an agent can react in real-time and according to the environment. Nevertheless, when necessary, due to the complexity of the objective to be achieved, intelligent agents are integrated into distributed systems called *Multi-Agent Systems*, which are made up of a sum of autonomous but linked and collaborating agents.

Definition 4.2.2. ***(Multi-Agent System)***
A multi-agent system is a community of autonomous agents evolving in a common environment, according to occasionally complex modes of cooperation, competition or even conflict, in order to achieve an overall objective.

The key point of multi-agent systems lies in the formalization of coordination between agents. Therefore, when developing a multi-agent system, some features should be taken into consideration. The first thing to consider is the mechanism of agent *decision*. It is about the perceptions, representations, and actions of agents as well as the way they break down their goals and tasks. Next, the *control* of agents

should be defined. This consists in the relations and coordination between agents. The coordination can be described as cooperation to accomplish a common task or as negotiation between agents with different interests. Finally, it is important to determine *communication* between agents that is the type, syntax, and protocol used to exchange messages between agents.

4.2.2 Motivation of Using a Multi-Agent System

In order to facilitate the interpretation of logged events, and determine their compliance with security rules, key concepts in logs are extracted. The extracted concepts provide the information concerning what object was accessed by which subject through which action and at what time (*c.f. Chapter 3*). However, when using a significant security policy like in the case of ABAC, more attributes should be injected in logs to add more semantics. Thus, rendering semantic logs will help the experts in analyzing and explaining users' actions as it allows causal interpretation. In contrast, information sources are usually distributed throughout the organization's systems, and the collection and integration of this information is not trivial. Moreover, the number and variety of data sources and services increases as new applications are being developed, and the availability and reliability of information services are constantly changing. Besides, the same piece of information can be accessible through a variety of different sources, and this information is prone to updates.

For all the reasons discussed above, deploying a multi-agent system appeared to be suitable to achieve the log semantic enrichment process for its ability to locate, access, and gather information from various data sources. Nevertheless, since the information to be collected to complement logs can be updated at any time, it is necessary that the multi-agent system goes beyond information gathering, to include temporal verification. The importance of temporal verification of attributes' values is discussed next.

4.2.3 Criticality of Policy Temporal Compliance

Until this point, we have established that logs need to be semantically enriched to be able to perform an a posteriori access control and detect violations. Yet, in an information system, data sources are usually not static. There is always an administrative action that can change the attributes assigned to subjects or objects at any time. Besides, contextual conditions evolve depending on the situation. Therefore, the variation of attributes and attributes' values assignments over time leads to changes in the applicability of the defined rules that state the permissions assigned according to a group of attributes.

This highlights the importance of the temporal aspect, when realizing investigations, to ensure that the right attributes that permit the access were in place *at the time* of the execution of the action. This can be the case of a doctor whose status had changed from "visiting" to "permanent", and where he had fewer privileges when he was "visiting". Therefore, it is important to check the status that he had when he performed the access.

That being said, we broaden the a posteriori access control to have a finer granularity that includes temporal verification. We define the problem of policy temporal compliance as follows.

Definition 4.2.3. ***(A Posteriori Policy Temporal Compliance)***
Verifying the a posteriori policy temporal compliance consists in checking if the required condition for an access to be authorized held ***at the time*** *of the access.*

We thus formalize this temporal verification using the Event Calculus, which we explained next.

4.2.4 Event Calculus

In this section, we provide some background on the Event Calculus, and show how it can be modelled using Semantic Web technologies, based on which we built our policy temporal compliance framework.

Background on the Event Calculus

The Event Calculus (EC) is a logical language for representing and reasoning about events and their effects. The authors in [185], described it as "a logical mechanism that infers what's true when given what happens when and what actions do". It is defined in many-sorted first order logic, which is an extension to first order logic that provides the notion of types. Thus, the presence of typing makes it possible to specify semantics through logic.
Moreover, the EC has undergone several variations [138] from its first occurrence [119]. In this work we use the form presented in [185], that consists of: (1) a set of event types or actions (2) a set of fluents, that is a set of properties which values can change over time (and can be true or false) (3) a set of time points. These three elements are essential and are used through predicates that constitute the language. We represent the most commonly used predicates in *Table 4.1*.

Table 4.1: Event Calculus Basic Predicates

Predicate	Meaning
Initiates(e,f,t)	if event *e* is executed at time *t*, fluent *f* is true after *t*
Terminates(e,f,t)	if event *e* is executed at time t, fluent *f* is false after *t*
Happens(e, t)	event *e* occurs at time *t*
HoldsAt(f,t)	fluent *f* holds at time *t*
Clipped(t1,f,t2)	fluent *f* is terminated between times *t1* and *t2*

An EC domain description consists of an axiomatization, observations of world properties, and a narrative of known world events; hence, given a domain description, various types of commonsense reasoning can be performed such as the deductive, inductive, and abductive reasoning. Deduction uses the description of the system behaviour together with the history of events occurring in the system to derive the fluents that will hold at a particular point in time. Induction aims to derive the initial state given a set of events and fluents' states at specific timepoints, while abduction tend to determine the sequence of events that need to occur given the system's description and a set of fluents that will hold at a specific time. In this work, we use

the deductive reasoning supported by the EC, as the history of events occurring in the system (logs) is used to derive the fluents that will hold (permissions). Therefore, EC is able to represent cause and effect as it treats time-varying properties (fluents) and events as objects so that, using axioms, statements can be made about the truth values of properties and the occurrences of events at specific timepoints. These axioms can be formed by relating the various predicates together to describe how events and fluents interact.

We now require a suitable collection of axioms relating the various predicates together:

$$\begin{aligned} &Happens(e, t_1) \wedge Initiates(e, f, t_1) \wedge (t_1 < t) \wedge \neg Clipped(t_1, f, t) \\ &\rightarrow HoldsAt(f, t) \end{aligned} \tag{4.1}$$

$$\begin{aligned} &\exists e, t[Happens(e, t) \wedge (t_1 \leq t < t_2) \wedge Terminates(e, f, t)] \\ &\longleftrightarrow Clipped(t_1, f, t_2) \end{aligned} \tag{4.2}$$

Axiom (4.1) indicates that a fluent is true at time t if it has been made true in the past and has not been made false in the meantime. The predicate *Initiates* introduces the event, that activates the fluent, at the time of its execution. For instance, assigning the role Doctor to a user, leads to the user having the role Doctor. This can be expressed using Initiates as *Initiates(setRole(user,Doctor), role(user,Doctor), t)*. Similarly, *Terminates(removeRole(user,Doctor), role(user, Doctor), t)*, indicates that removing the role Doctor of a user terminates the fact of that user being a Doctor.

Moreover, the *Clipped* predicate presented in *(4.2)*, states that an event's occurrence terminates a fluent during an interval of time.

With respect to the above axioms of the simple EC, we define handful expressions on which the a posteriori temporal compliance will be based.

We introduce the predicate *Always* to indicate that a fluent's value will remain the same over time. Thus, a fluent f is always true if and only if it holds at any time t, as follows:

$$Always(f) \longleftrightarrow \forall\, t,\ HoldsAt(f,t) \tag{4.3}$$

That being said, if a fluent holds at time t, and at time t, the fact of being true always

causes another fluent to be true, then this latter is also true at time t:

$$\forall\, t,\ HoldsAt(f \rightarrow g,t) \wedge HoldsAt(f,t) \rightarrow HoldsAt(g,t) \tag{4.4}$$

In addition, saying that the conjunction of two fluents is true, is equivalent to saying that each fluent is true:

$$\forall\, t,\ HoldsAt(f \wedge g,t) \longleftrightarrow HoldsAt(f,t) \wedge HoldsAt(g,t) \tag{4.5}$$

It is worth mentioning that the events in EC can be natural events like lightning or accidental crash of a hard disk. Since we are dealing with access control, we shall consider, in the following, events that are caused by the execution of an action by a subject on an object.

Modelling the Event Calculus in SWRL

Event Calculus comes from a line of logic formalism for commonsense reasoning, which started with the Situation Calculus [164]. A discussion about implementing this latter using SWRL was put on the table in the Semantic Web research community [134]. From this discussion, it was deduced that it is possible to model the EC in SWRL. Therefore, the authors in [137] developed an ontology for a simplified version of the EC that deals with discrete time points, that is the Discrete Event Calculus (DEC). In this ontology, each of the *Fluent*, *Event*, as well as the Event Calculus' predicates *HoldsAt*, *Happens*, *Initiates*, *Terminates*, and *Clipped*, are represented as Classes, and are related with the properties *hasEvent*, *hasFluent*, and *hasTime*. Moreover, they expressed some DEC axioms, according to SWRL.

Modelling the EC predicates as Classes is justified by the fact that SWRL predicates do not support having more than two attributes, while some of the formers require more. Thus, we adapted their proposal so that it suits our case of the a posteriori access control. The major difference between their work and ours, is the way in which we apply the EC and compute the holding fluents. In their study, they reason over events as they occur progressively, meanwhile we are checking the state of the fluents a posteriori, which means that we will not wait for the occurrence of an event since the

events happened in the past.

Furthermore, to explain this modelization, we recall the *"reification"* or *"objectification"* approach that asserts that it is always possible to go back to binary association types from an n-ary association type (n>2), as follows: (1) replace the n-ary association type by an entity type and assign an identifier to it, and (2) create binary association types between the new entity type and all entity types of the collection of the old n-ary association type. Thus, it permits to take advantage of the richer semantics of the entity-relationship model to re-express the semantics of the *n*-ary relation:

Theorem 4.2.1. ***(Reification)***
Let $R(a_1, a_2, ..., a_n)$ *be a n-ary relation.*
Reifying R consists in creating a unary relation RE(e), and n binary relations $RA_1(E,a_1)$, ..., $RA_n(E,a_n)$, *which fulfill the following axiom:*
$\forall\ a_1, \forall\ a_2, \ldots, \forall\ a_n, R(a_1, a_2, \ldots, a_n) \longleftrightarrow \exists\ e, RE(e) \wedge RA_1(e,a_1) \wedge RA_2(e,a_2) \wedge \ldots \wedge RA_n(e,a_n)$.

In the case of the EC, e will be a created individual of the class representing an EC predicate (*e.g Happens, Initiates, Terminates, etc.*), a_i will be individuals of the classes representing the components of the EC (*Fluent, Event, and Time*), and RA_i the properties relating e to a_i. For example, the ternary relation *Initiates(e,f,t)* will be represented in SWRL as *Initiates(?initiates)* $\wedge$ *hasEvent(?initiates,?event)* $\wedge$ *hasFluent(?initiates,?fluent)* $\wedge$ *hasTime(?initiates,?time)*.
It has been also proven that this translation preserves semantics in [54].

Another reason to choose this translation is the lack of support of negation as failure in OWL and SWRL . The only way to express this latter in OWL/SWRL is to use classical negation, by defining the complement of the predicate (e.g. HoldsAt) as an OWL class (e.g. NotHoldsAt).

The interpretation of *(4.1)* in SWRL is as follows:

```
Happens(?happens) ∧ Event(?e) ∧ hasEvent(?happens,?e)
∧ hasTime(?happens,?t1) ∧ Initiates(?initiates) ∧
hasEvent(?initiates,?e) ∧ hasFluent(?initiates,?fluent)
∧ hasTime(?initiates,?t1) ∧ NotClipped(?notClipped) ∧
hasStartTime(?notClipped,?t1) ∧ hasEndTime(?notClipped,?t)
∧ hasFluent(?notClipped,?fluent) ∧ swrlb:lessThan(?t1,?t)
∧ swrlx:makeOWLThing(?holdsAt,?t) → HoldsAt(?holdsAt) ∧
hasFluent(?holdsAt,?fluent) ∧ hasTime(?holdsAt,?t)
```

In a similar way, the axiom *(4.2)* defining the Clipped predicate can be expressed in SWRL as follows:

```
Happens(?happens) ∧ Event (?e) ∧ Terminates(?terminates)
∧ Clipped(?clipped) ∧ hasTime (?happens, ?t) ∧
hasStartTime(?clipped, ?t1) ∧ hasEndTime (?clipped, ?t2)
∧ hasTime(?terminates, ?t) ∧ hasEvent (?terminates, ?e) ∧
hasFluent (?terminates, ?f) ∧ swrlb:lessThan (?t1, ?t) ∧
swrlb:lessThan (?t, ?t2) → hasFluent(?clipped, ?f)
```

The rest of the expressions are given along with the following sections.

4.3 Modelling the Security Policy with ABAC and OWL

In the a posteriori access control, the security policy constitutes a reference that is consulted to decide whether a logged event is a violation or not. Therefore, it is essential to have a formal representation of the security policy.

We chose to represent the security policy according to ABAC since it allows more flexibility than any other access control model. Its key benefit is that it grants access based on the attributes of each system component. Thus, complex rules can be defined (*e.g., access is allowed at specific hours*), and the multiplicity and variety of attributes makes an ABAC system able to represent any other access control model (*c.f. Section 2.1.1*).

In addition, we have provided in *Section 2.3*, the advantages offered by modelling the security policies in OWL, and presented some distinguished works in this area. Therefore, we got inspired from [73] and [186] to formalize our ABAC policy in OWL.

To start with, each of the *Subject*, *Object*, *Action*, and *Context* are defined as *Classes*, and the corresponding *Attributes* are defined as *Properties*.

```
Subject a owl:Class  subjectAttribute a owl:ObjectProperty
Object a owl:Class   objectAttribute a owl:ObjectProperty
Action a owl:Class   actionAttribute a owl:ObjectProperty
Context a owl:Class  contextAttribute a owl:ObjectProperty
```

```
subject a rdfs:Property, owl:FunctionalProperty;
rdfs:domain Action;
rdfs:range Subject.

object a rdfs:Property, owl:FunctionalProperty;
rdfs:domain Action;
rdfs:range Object.

context a rdfs:Property, owl:FunctionalProperty;
rdfs:domain Action;
rdfs:range Context.
```

The definition of the attributes is very application-specific. Therefore, we cannot generalize their definition in the ABAC ontology. We can only say that they are defined as separate classes and their relationship with their relative domain (Subject, Object, Action, or Context) are defined as sub-properties of the above defined attributes properties. For example, if we have *Role* as a *Subject Attribute*, *Owner* as an *Object Attribute*, and *Type* as an *Action Attribute* they will be defined as follows:

```
role a owl:ObjectProperty;
rdfs:subPropertyOf subjectAttribute;
```

```
   rdfs:domain Subject;
   rdfs:range abac:Role.

owner a owl:ObjectProperty;
rdfs:subPropertyOf objectAttribute;
   rdfs:domain Object;
   rdfs:range abac:Owner.

type a owl:ObjectProperty;
rdfs:subPropertyOf actionAttribute;
   rdfs:domain Action;
   rdfs:range abac:Type.
```

Moreover, attribute values are created by defining individuals of their corresponding classes, such as *Role*, *Owner* and *Type*, and their subclasses in case they exist.

Traditionally, an ABAC system requires a proper attribute assignment to ensure the appropriate accesses. Nevertheless, to fulfill the essential high-level access control requirements, certain constraints specification on attribute values may be needed. This problem of constraint specification has been known in RBAC as *Separation of Duty (SOD)* that is often characterized as *Static Separation of Duty (SSOD)* and *Dynamic Separation of Duty (DSOD)* [80, 99]. *SSOD* constraints specify pairs of roles where any subject can only have one of the pair as a possible role, while *DSOD* constraints hold between two roles when no subject can have both simultaneously active. For instance, an employee in a hospital cannot have the roles “Doctor” and “Nurse” at the same time.

Constraint specification in ABAC is more complex than that in RBAC since there are multiple attributes. These constraints can exist among different values of a set-valued attribute and/or on values across different attributes [27]. For example, the constraint can represent a mutual exclusion conflict between two values, a cardinality constraint on mutual exclusion, a precondition constraint, etc. Without loss of

generality, these constraints can be expressed in OWL by defining the appropriate object and datatype properties. For example, the properties representing *SSOD* and *DSOD* (roles are defined as attributes in ABAC), can be defined as below:

```
ssod a owl:ObjectProperty;
owl:symmetricProperty; owl:TransitiveProperty;
   rdfs:domain abac:Role;
   rdfs:range abac:Role.

dsod a owl:ObjectProperty;
owl:symmetricProperty; owl:TransitiveProperty;
   rdfs:domain abac:Role;
   rdfs:range abac:Role.
```

In contrast, to enforce the security policy, we model its defined rules according to SWRL (*c.f. Section 2.4.3*). For instance, the following SWRL rule defines the policy rule "*A patient can view his own medical record*".

```
Action(?a) ∧ subject(?a,?s) ∧ object(?a,?o) ∧ type(?a,View)
∧ role(?s,Patient) ∧ oType(?o,MedicalRecord) ∧ owner(?o,?s)
→ isPermitted(?a)
```

4.4 Multi-Agent Based Policy Temporal Compliance Framework

The accuracy of the results generated by a policy violation detection mechanism lies on the quality of linking log traces to security rules; hence, we distinguish two inevitable components of the a posteriori access control that are: *logs* and the *security policy*. Moreover, we have previously discussed the need to semantically enrich log information as well as to verify the temporal validity of the resulted information (after enrichment). Therefore, our policy compliance mechanism will answer the following:

(1) For a logged event, do the user and object have the right attributes and values, and is the action executed in the right context?
(2) If they have ever had the right attributes, did they have them at the same time of the access?
Therefore, we can distinguish two approaches. The first one consists in taking all the logs and translating them into SWRL facts. This approach is not satisfactory, since it requires a lot of time to translate, which is costing, in addition to the need of loading all the facts into memory. The second approach, that we adopted, goes through a semantic mediator and a multi-agent approach to get the necessary information as and when it is needed.

As we presented how to extract information from multiple log sources in *Chapter 3*, we present, in the following, our multi-agent based policy temporal compliance framework.

4.4.1 Multi-Agent System Architecture

The goal of the proposed multi-agent system is to gather the needed attributes from different organizational data sources, and check if their temporal assignment is conform with the security policy or not. The proposed multi-agent system architecture is depicted in *Figure 4.1*.

We thus, distinguish four types of agents that we present in details in the following subsections:

- **Policy Agent** handles the rules defined in the security policy. It is the one responsible of providing the attributes to be fetched to the mediator agent.
- **Mediator Agent** is the *maestro* of the whole information gathering process. Once it gets the attributes from the policy agent, it orchestrates all the exchanged messages with the agents.
- **Data Source Agent** retrieves information from a specific data source.
- **Event Calculus Agent** verifies the temporal conditions defined in the security policy using the Event Calculus.

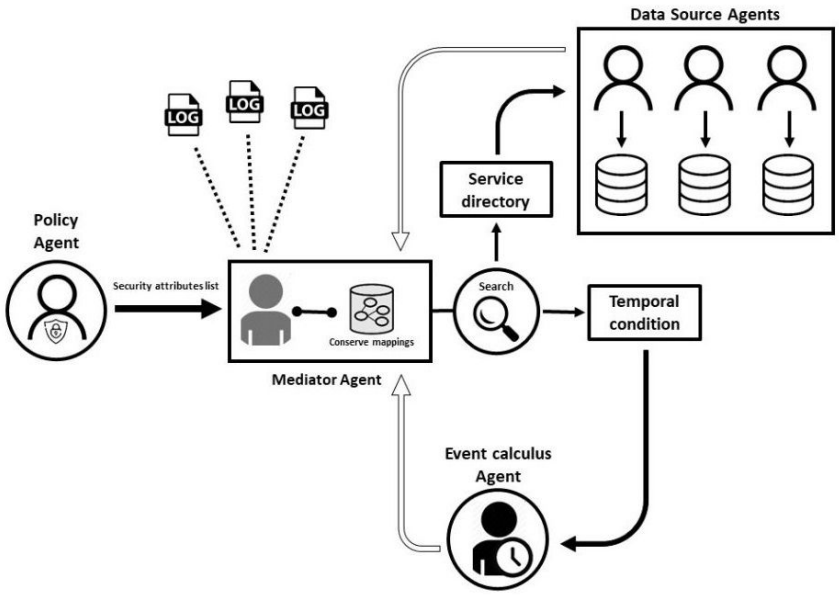


Figure 4.1: MAS Architecture

Before getting into the functioning of the system, we define some terms that we use along the explanation:

- ***Task***: is the action to be performed by an agent.
- ***Service***: is the kind of information that an agent provides.
- ***Request message***: is a message sent from an agent to another asking to perform an action. It is represented as *Request(T,S,R,m)*, where *T* is the requested task, *S* is the Sender Agent, *R* is the Receiver Agent, and *m* is the content of the message.
- ***Inform message***: is a message sent from an agent providing an information or responding to a request message. It is represented as *Inform(S,R,m)* where *S*,*R*, and *m* are as defined above.

4.4.2 Multi-Agent System Functioning

In this section, we present the functioning of each agent of the proposed architecture.

The Policy Agent

The ***Policy Agent (Po)*** is located on the policy side. Its mission is to deliver the list of the defined attributes in the security rules to the ***Mediator Agent***.

As shown in *Section 4.3*, security rules are modelled in SWRL. Thus, each rule has the form of *Condition* $\rightarrow$ *is-permitted(u,op,o)*. It is worth noting that *is-permitted(u,op,o)* is expressed in SWRL as *isPermitted(?action)*, where *?action* has the type *op* and is related to *u* and *o* with the predicates *subject* and *object*, respectively.

Moreover, the condition is composed of the subject, object, action, and their respective attributes and attributes' values (since rules are modelled according to ABAC). Therefore, ***Po*** parses the defined SWRL rules to construct a list of Subject Attributes, Object Attributes, and Environmental Attributes (context) to be verified. For each predicate representing an attribute defined in the SWRL rule, the agent gets the domain and range of the predicate and associates the defined values to the corresponding classes. In consequence, the constructed list contains tuples of the form <*att, v, h*>, where *att* is the attribute's name, *v* is the attribute's value, and *h* is the attribute's holder (e.g., subject, object, etc.).

In contrast, we consider in this chapter that the expression of the security policy is static and do not go through changes over time. In other words, the rules in the security policy are defined once and for all, meanwhile their application may depend on contextual conditions. This hypothesis is relaxed in the next chapter. Therefore, every security rule can be expressed as:

$$Always(Condition \rightarrow is\text{-}permitted(u,op,o)) \tag{4.6}$$

With respect to the rule defining *Always* (*axiom (4.3)*), an action *op* done by a user *u* on an object *o* at a specific time *t* is considered as permitted if the required condition held at that same time t. We recall that in ABAC, the condition consists in the attributes and attributes' values acquired by the entities presented in the access.

The Mediator Agent

The ***Mediator Agent (Med)*** is settled in the semantic mediator that we presented in *Chapter 3*. We recall that this mediator extracts information from logs in the form of a tuple <*Subject, Action, Object, Timestamp*>. Therefore, we assimilate a logged event to an event in the Event Calculus that follows *Happens(e, t)*, and we denote it as e = *(u,op,o)*, where *op* is an action (operation) that was executed by a user u on an object o, and that happened at a certain time t.

Moreover, the occurrence of an event can be represented in SWRL as follows:

```
Happens(?happens) ∧ Action(?e) ∧ type(?e,?op) ∧ subject(?e,?u)
∧ object(?e,?o) ∧ hasEvent(?happens,?e) ∧ hasTime(?happens,?t)
```

Note that *Action* is a subclass of *Event*.

Once the list of attributes is received, ***Med*** starts the semantic enrichment process. Its main goal is to get these attributes' values and timestamps, relatively to the information extracted from logs, to verify their compliance with the policy.

It is worth mentioning that we consider that the logs contain at least one element, from which we can get the security attributes defined in the security policy. Moreover, to detect the type of the extracted values, for example, if the subject's extracted value corresponds to a UserID, HostName, IP, etc., we used regular expressions (regex) and a dictionary-based classifier [169].

Therefore, to search for a specific attribute, ***Med*** searches in a service directory, where several other agents registered the services that they provide, and identifies the agent to which it should send a request message to get the corresponding attribute's value and timestamp. The content of the corresponding messages is a tuple <*att, v, ht, hv*>, where *att* and v are, respectively, the attribute's name and attribute's value to search for, that were received from ***Po***, *ht* is the identified type of the holder using regex as discussed above (e.g., MID), *hv* is the value of the holder extracted from logs (e.g., 9000000085). Thus, the message sent from ***Med*** to the identified agent can be formulated as follows: "*At what time did the ht with hv have the value v for the attribute att?*".

Data Source Agents

In a real organization, not all the information is stored in one place. It can have many databases that can have the same or different type of information. That is why we consider having many data sources, each one represented by a ***Data Source Agent (DS)*** that registers the service it provides in a service directory when it joins the multi-agent system.

When a ***DS*** receives a request, it gets the corresponding information. For instance, if the data source is an SQL database, the agent will execute SQL queries, and replies to the mediator agent with an inform message containing the requested information. Moreover, we consider that the ***DS*** has access to the history logs of the data source, and that it knows the events responsible for assigning and removing an attribute's value. It will search then for these events, that are related to the extracted log elements, and their timestamps, and send them back to ***Med***. Thus, the content of the sent message is a list of tuples <*e*,*t*> where *e* = *(u,op,o)* is the initiating/terminating event that activates/deactivates the value of the requested attribute and *t* its timing.
It must also be pointed out that agents may have different vocabularies. To resolve this heterogeneity, mappings can be established between the different concepts handled by the different agents (e.g., equivalence between two different entities handled by two different agents) [208].

The Event Calculus Agent

Once ***Med*** has collected all the attributes' values and the times of their assignment/removal, it sends them in an inform message to the ***Event Calculus Agent (EC)***, so that it can assess policy temporal compliance.

The main goal is to deduce a violation when a non-permitted access is logged (is done); hence, verifying the following:

$$Happens((u,op,o),t) \wedge \neg\, HoldsAt(is\text{-}permitted(u,op,o),t) \rightarrow violation(u,op,o) \tag{4.7}$$

Expression (*4.7*) can be expressed in SWRL as follows:

```
Happens(?happens) ∧ Action(?e) ∧ type(?e,?op) ∧ subject(?e,?u)
∧ object(?e,?o) ∧ hasEvent(?happens,?e)∧ hasTime(?happens,t)
∧ NotHoldsAt(?notholdsAt) ∧ isPermitted(?e) ∧
hasFluent(?holdsAt,?e) ∧ hasTime(?holdsAt,?t) → Violation(?e)
```

The event *(u,op,o)* is the one extracted from logs by the semantic mediator. However, to check if it was permitted or not at the time of its occurrence we should check if the required condition was holding at that same time as in expression (*4.6*).

In ABAC, the condition consists in having the right subject attributes, object attributes, and environmental attributes (context), with the right values. Since these attributes may evolve over time, we consider each one of them as a fluent. Therefore, each property representing an attribute in the antecedent of a SWRL rule will be mapped to a fluent.

Let m be a function that maps each pair of *attribute-value* to a fluent:

$$m\text{: } ATT * Dom(ATT) \longrightarrow Fluents$$
$$x.att_i = v_i \longrightarrow f_i$$

where *ATT* is the set of all attributes, *Dom(ATT)* is the set of all possible values that an attribute can take, $x \in$ {Subject, Object, Environment}, att_i is the attribute name, and v_i its value.
Hence, the conjunction of all the fluents f_i constitutes the final condition to be verified, that is also a fluent. We define the condition fluent as:

$$f_{cond} = \bigwedge_{i=1}^{n} f_i$$

where n is the total number of the required access attributes.
Therefore, to verify if the condition holds at time t, we need to check
$HoldsAt(f_{cond}, t) \equiv HoldsAt(f_1 \wedge f_2 \wedge ... \wedge f_n, t)$, by applying (*4.5*).

To express this conjunction of fluents in SWRL, we consider having two disjoint

subclasses, *SuperFluent* and *SubFluent*, of the class *Fluent*, that represent f_{cond} and f_i respectively, and are related with the property *hasSubFluent*. Thus, for each condition fluent, we generate an individual of the class *SuperFluent* that is related to its sub-fluents with *hasSubFluent(?f,?*f_i*)*.

Next, once the sub-fluents are identified, we need to check if they hold at access time *t* according to expression (*4.1*). It is worth mentioning that fluents that necessitate two arguments, have the relations *hasDomain* and *hasRange* to refer to their arguments. For instance, the subject attribute *role(?u,Doctor)* is considered as a fluent, and when reasoning in EC, it is expressed in SWRL as: *Role(?fluent)* $\wedge$ *hasDomain(?fluent,?u)* $\wedge$ *hasRange(?fluent,Doctor)*.

Continuing, the occurrence of events that control the state of the fluent will initiate/terminate the corresponding fluent (e.g., *setRole(?u,Doctor)* will initiate the fluent *role(?u,Doctor)*). Therefore, the intiation of an attribute fluent can be expressed in SWRL as follows:

```
Happens(?happens) ∧ InitiatingEvent(?e) ∧ hasDomain(?e,?d) ∧
hasRange(?e,?r) ∧ hasEvent(?happens,?e) ∧ hasTime(?happens,?t)
∧ swrlx:makeOWLThing(?initiates,?e) ∧ SubFluent(?fluent)
∧ Attribute(?fluent) ∧ hasDomain(?fluent,?d) ∧
hasRange(?fluent,?r) → Initiates(?initiates) ∧
hasEvent(?initiates,?e) ∧ hasFluent(?initiates,?fluent) ∧
hasTime(?initiates,?t)
```

Similarly, the termination of an attribute fluent can be expressed in SWRL as follows:

```
Happens(?happens) ∧ TerminatingEvent(?e) ∧ hasDomain(?e,?d) ∧
hasRange(?e,?r) ∧ hasEvent(?happens,?e) ∧ hasTime(?happens,?t)
∧ swrlx:makeOWLThing(?terminates,?e) ∧ SubFluent(?fluent)
∧ Attribute(?fluent) ∧ hasDomain(?fluent,?d) ∧
hasRange(?fluent,?r) → Terminates(?terminates) ∧
hasEvent(?terminates,?e) ∧ hasFluent(?terminates,?fluent) ∧
hasTime(?terminates,?t)
```

In contrast, the *Clipped* and *NotClipped* predicates were introduced in EC to deal with causal constraints; hence, it is necessary to support existential quantification and two way implication to translate these axioms into rules. Nevertheless, SWRL lacks existential quantifiers which makes it impossible to express (*4.1*), for example, in SWRL alone. Therefore, we couple SWRL with algorithms to have a correct implementation of the EC axioms. For instance, when a fluent is not clipped between its initiating and terminating event, the creation of a *NotClipped* instance is enforced by an algorithm. The dedicated alogrithm uses SQWRL queries to retrieve the timestamps of the initiating and terminating events of a fluent. For example, to get the initiates events and timestamps of the fluents, more precisely sub-fluents, the below SQWRL query is used:

```
Initiates(?initiates) ∧ hasEvent(?initiates,?e) ∧
hasFluent(?initiates,?f) ∧ SubFluent(?f) ∧ hasDomain(?f,?d)
∧ hasRange(?f,?r) ∧ hasTime(?initiates,?t) →
select(?initiates,?e,?f,?d,?r,?t)
```

The retrieval of the terminates events are done in a similar way. After the above elements are retrieved, the function *AssertNotClippedStatement* is called to create a *NotClipped* instance. The *AssertNotClippedStatement* is similar to the *AssertHoldsForStatement* given in *Section 5.6*. Now that all the needed axioms are asserted, the *HoldsAt* statements can be deduced as in (*4.1*).

After checking if each sub-fluent holds at t or not, we need to check if the final fluent (the conjunction of all sub-fluents), holds at t. To do so, we used the following SQWRL query:

```
HoldsAt(?holdsAt) ∧ hasFluent(?holdsAt,?f) ∧
SubFluent(?f) ∧ sqwrl:makeSet(?s, ?f) ∧ sqwrl:groupBy(?s,
?holdsAt) ∧ SuperFluent(?fl) ∧ hasSubFluent(?fl,?fs)
∧ sqwrl:makeSet(?s2,?fs) ∧ sqwrl:groupBy(?s2,?fl) ∧
sqwrl:contains(?s,?s2) → sqwrl:select(?holdsAt,?fs)
```

This latter, constructs two sets of sub-fluents, one for each generated *holdsAt*

individual, and one for each *SuperFluent* defined initially. After that, it compares the two obtained sets. If the set of the initially defined fluents is contained in the generated fluents set, then the query returns a result and a new *HoldsAt* individual is created with the corresponding *SuperFluent* and access *time* associated. If the query result is empty, it means that at least one of the sub-fluents does not hold at t, leading to the creation of a *NotHoldsAt* individual associated with the *SuperFluent* and access *time*.

The utility of using SQWRL queries and complementing SWRL with algorithms for the creation of the *NotHoldsAt* and *NotClippedAt* individuals is to assure negation. It is known that OWL and SWRL are based on the open-world assumption, thus, the truth of facts cannot be determined unless explicitly stated. Therefore, it has been demonstrated in [145] that negation-as-failure can be implemented on top of purely open-world systems using queries.

Finally, expression (*4.6*) can be expressed in SWRL as follows:

```
HoldsAt(?holdsAt) ∧ hasFluent(?holdsAt,?f) ∧
hasTime(?holdsAt,?t) ∧ SuperFluent(?f) ∧ isRelatedTo(?f,?e)
∧ type(?e,?op) ∧ subject(?e,?u) ∧ object(?e,?o) ∧
swrlx:makeOWLThing(?holdsAt2,?holdsAt) → HoldsAt(?holdsAt2)
∧ isPermitted(?e) ∧ hasFluent(?holdsAt2,?e) ∧
hasTime(?holdsAt2,?t)
```

The *isRelatedTo* property is added to the *SuperFluent* representing the condition so that we can relate the collected attributes to the occurred event; hence, deduce if the event is permitted or not.

Furthermore, verifying the respect of constraints, particularly separation of duty, using the Event Calculus is very relevant since it reasons over time by nature. Therefore, the *Static Separation of Duty* can be verified by checking if a single user did not have two roles that are related with a *ssod* constraint at any time. This is done by using two different time variables *?t* and *?t2* as follows:

```
HoldsAt(?holdsAt) ∧ Role(?r) ∧ hasFluent(?holdsAt,?r) ∧
hasDomain(?r,?d) ∧ hasRange(?r,?x) ∧ hasTime(?holdsAt,?t)
∧ HoldsAt(?holdsAt2) ∧ Role(?r2) ∧ hasDomain(?r2,?d) ∧
hasRange(?r2,?y) ∧ ssod(?x,?y) ∧ hasTime(?holdsAt2,?t2) ∧
SuperFluent(?f) ∧ hasSubFluent(?f,?r) ∧ isRelatedTo(?f,?e)
∧ swrlx:makeOWLThing(?notHoldsAt) → NotHoldsAt(?notHoldsAt)
∧ isPermitted(?e) ∧ hasFluent(?notHoldsAt,?e) ∧
hasTime(?notHoldsAt,?t)
```

As for checking the *Dynamic Separation of Duty,* it is about verifying if a user had two roles with a *dsod* constraint at the same time by using a single variable *?t*:

```
HoldsAt(?holdsAt) ∧ Role(?r) ∧ hasFluent(?holdsAt,?r) ∧
hasDomain(?r,?d) ∧ hasRange(?r,?x) ∧ hasTime(?holdsAt,?t)
∧ HoldsAt(?holdsAt2) ∧ Role(?r2) ∧ hasDomain(?r2,?d) ∧
hasRange(?r2,?y) ∧ dsod(?x,?y) ∧ hasTime(?holdsAt2,?t) ∧
SuperFluent(?f) ∧ hasSubFluent(?f,?r) ∧ isRelatedTo(?f,?e)
∧ swrlx:makeOWLThing(?notHoldsAt) → NotHoldsAt(?notHoldsAt)
∧ isPermitted(?e) ∧ hasFluent(?notHoldsAt,?e) ∧
hasTime(?notHoldsAt,?t)
```

It is also worth to mention that since the policy compliance is checked a posteriori and not in real time, the verification is done rule by rule, and the decision of whether there is a violation or not is computed once all the attributes and timestamps are gathered. Moreover, the violation of constraints such as *Separation of Duty* is also detected when performing the a posteriori access control.

4.5 Use Case

Always in the medical field, we provide in this section a use case to show the utility of our approach.

Consider the following medical rule that needs to be verified:

"A doctor may create a prescription during an office visit".

This rule can be expressed in SWRL as follows:

Action(?a) $\wedge$ *type(?a,create)* $\wedge$ *subject(?a,?u)* $\wedge$ *role(?u,Doctor)* $\wedge$ *object(?a,?o)* $\wedge$ *oType(?o,Prescription)* $\wedge$ *context(?a,?w)* $\wedge$ *cType(?w,OfficeVisit)* $\rightarrow$ *isPermitted(?a)*.

Moreover, we consider that we extracted, using the semantic mediator, the following event e_1 from the logs:

Subject	**Action**	**Object**	**Timestamp**
9000000003	CREATE	PRE35876	2019-07-22 14:59:04

We also consider that there are two Data Source agents $\boldsymbol{DS_1}$ and $\boldsymbol{DS_2}$ that provide the *Role* of a *Medical ID (MID)*, and the *Type* of a *Resource ID*, respectively.

Illustrating the steps provided in *Section 4.4.2*, the list sent from ***Po*** to ***Med*** contains the attributes *role* of the *Subject*, *type* of the *Object*, and the *type* of the *Context* in which the action should be done at every time, in addition to their respective values. The content of the message sent from ***Po*** to ***Med*** is shown in *Figure 4.2*.

```
(sequence (Attribute :name role :value Doctor :holder Subject) (Attribute :name oType :value Prescription
:holder Object) (Attribute :name cType :value OfficeVisit :holder Context))
```

Figure 4.2: List of Attributes sent by ***Po*** to ***Med***

When ***Med*** receives the list of attributes, it sends request messages to $\boldsymbol{DS_1}$ and $\boldsymbol{DS_2}$ to get the timestamps of the events responsible for assigning and/or removing the Role and Type of 9000000003 and PRE35876, respectively. $\boldsymbol{DS_1}$ and $\boldsymbol{DS_2}$ are identified after searching in a service directory. These messages have the following forms: *Request(*`SearchAttributeTime`*, Med,* DS_1*, (Role, Doctor, MedicalID, 9000000003))* and *Request(*`SearchAttributeTime`*, Med,* DS_2*, (oType, Prescription, ResourceID, PRE35876))*. We provide an example of the representation of these messages in the Agent Communication Language (ACL) [165] in *Figure 4.3*.

```
(REQUEST
  :sender    ( agent-identifier  :name Med@IP1/JADE   :addresses (address))
  :receiver     (set  (  agent-identifier :name  DS1@IP2/JADE  ) )
  :content    "( (action (agent-identifier  :name DS1@IP2/JADE) (SearchAttribute
:type  role   :holder (Holder :type MID :value \"9000000003\") ) ) )"
  :language   fipa-sl  Policy-Ontology )
```

Figure 4.3: Agent message in ACL

Continuing, $\boldsymbol{DS_1}$ will search for the timestamps of the events *setRole(9000000003, Doctor)* and *removeRole(9000000003, Doctor)*, if any, and replies to ***Med*** with an inform message as follows: *Inform(DS_1, Med, (setRole(9000000003,Doctor), 2019-05-16 10:34:21))*. Similarly, $\boldsymbol{DS_2}$ will reply to ***Med*** with the timestamps of the events concerning the type of the Object: *Inform(DS_2, Med, (setType(PRE35876, Prescription), 2019-07-22 14:59:04))*.

Moving on to the contextual condition, it can be looked up in a similar way, as our approach is generic. Normally, its activating and deactivating events appear in the application logs; hence, the semantic mediator is used to look for them homogeneously. However, ***Med*** does not have an a priori knowledge of them. Thus, it will solicit ***EC***, where they are defined.
Med sends a request message to ***EC*** asking it for the initiating and terminating events of an office visit.
Considering that in an EHR application, the *office visit* holds from the time of its creation, till the time it is saved, we consider that the activating and terminating events of an office visit are *create office visit* and *save office visit*, respectively. After that, ***Med*** queries the logs to get the timestamps of these events as in *Chapter 3*.

At this point, ***Med*** has collected all the attributes values and the time of their assignment/removal. Now that all the condition inputs are ready, ***Med*** sends them in an inform message to ***EC***, so it can assess policy compliance according to expression (*4.7*). In this respect, the mapped fluents (sub-fluents) are f_1 = *role(?u,Doctor)*, f_2 = *oType(?o,Prescription)*, *and* f_3 = *cType(?w,OfficeVisit)*. Therefore, verifying (*4.7*) leads to

verifying (*4.6*) to see if the logged event is permitted or not at the time it was done. Furthermore, (*4.6*) consists in validating if the value of the attribute *Role* of the subject, the value of the attribute *Type* of the object, and the contextual condition *office visit* hold at *2019-07-22 14:59:04* as in (*4.1*).
Supposedly that f_1 and f_2 were true at the time of the access, and *2019-07-22 15:32:45* and *2019-07-22 16:05:18* are the timestamps at which the user *9000000003* has created and saved the office visit *OFF91383*, respectively, the contextual condition *office visit* was not holding at *2019-07-22 14:59:04*, since it was started after the creation of the prescription, leading to the detection of a violation.

4.6 Implementation And Evaluation

4.6.1 Implementation

To develop our multi-agent system, we used JADE in Java [20]. The adopted language for exchanging messages between agents is the Agent Communication Language (ACL) [165], and the content of the messages are ontology objects. Moreover, we used Protege 5.2.0 [167] as an environment for developing the policy ontology according to ABAC, as well as the Event Calculus ontology. We also used the OWL API [92] to parse SWRL rules and identify the classes and predicates that are used in, and SWRL API [151] to infer the EC rules. It is also worth to mention that we converted the timestamps into discrete time points in the system's time zone, to resolve time zones heterogeneity, using the Java APIs for Date and Time [102]. Moreover, a Windows machine was used with Intel(R) Core i5-7200U CPU at 2.7 GHZ, and all the agents were functioning on the same machine, while considering they are virtually on different ones.

4.6.2 Evaluation

The key concepts that we pertained to evaluate our violation detection mechanism are capability and performance. Therefore, we discuss, in the following, the capability metrics that are assured by our approach, as well as its performance.

Table 4.2: Capability Metrics

Metrics	Definition
Safety	checks if the access control policy leaks access permission to unauthorized principals.
Separation of Duty (SoD)	prevents error and fraud by ensuring that no conflict-of-interest assignments are assigned to a single subject.
Completeness	assures that each access request should be either accepted or denied by the access control policy.
Liveness	guarantees that there is no deadlock in which the system will wait forever for system events, and there is no livelock in which the access control model repeatedly executes the same operation forever.
Model-specific properties	are security properties that are specifically supported by various access control models.
Inconsistency detection	are conflicts between policy decisions that might occur.
Detection of redundant rules	checks if removing a rule does not change the behavior of the policy.

Capability

Normally, the capabilities of an access control policy verification model are described by a set of reference metrics. Therefore, [125] proposed some metrics to evaluate access control policy verification tools that we provide in *Table 4.2*.

The first metric that is usually considered is *safety*. It is about checking if the access control policy leaks access permission to unauthorized or unintended principals. Nevertheless, in the a posteriori access control, the environments are trustworthy and exceptional accesses are authorized depending on the context. Therefore, when detecting an access violation a posteriori, the system cannot be categorized as "*unsafe*"

as it might allow certain prohibited accesses on purpose. Therefore, evaluating *safety* in the a posteriori access control can be confusing as it inherently provides flexibility.

Moving on to the second metric, we have shown that our a posteriori access control framework includes the verification of *Separation of Duty*, that we expressed in OWL and the Event Calculus. It is worth noting that in a similar way, we can express other constraints relating to attributes' values in ABAC.

Moreover, *completeness* is a metric that is frequently examined. This latter assures that each access request should be either accepted or denied by the access control policy. It is evident that the response in our approach is boolean, since it consists in either a violation or not. When checking the compliance of a logged event with a security rule, all the attributes defined in that rule should be respected. If at least one required attribute did not hold at the time of the access, the corresponding rule in the security policy is considered to be violated. In consequence, our proposal is complete.

Furthermore, it is very important to assure *liveness*. Our approach guarantees it as we consider that all the attributes are logged somewhere (which is a security requirement), thus our policy compliance mechanism will neither wait nor repeat the same operation forever to find these attributes.

Besides, our approach is capable of *supporting any access control model*. For instance, ABAC can be replaced with RBAC or any other model, and the policy agent will do the job to inform of which elements should be searched for verification. Therefore, *model-specific properties* are respected, such as *availability*. The use of the Event Calculus allows us to check if a subject, for example, had the required attributes at a specific time. However, we consider the case of a static security policy and we treat the problematic of the evolution of this latter in *Chapter 5*.

Other interesting metrics are *inconsistency* and *redundancy*. In this work, we assume that the policy is free of conflict and redundancy. Thus, it is enough to have the logged event matching at least one rule in the security policy to decide that it is not a violation.

Performance

Implementing the multi-agent system to gather security attributes and check their temporal compliance allowed us to look into the feasibility of our approach, and to evaluate the execution time of the queries for performance issues. Even if some might consider that the access control is done a posteriori; hence, time is not of essence, a slow audit cycle is always undesired.

Moreover real organizations tend to minimize the number of attribute sources used in authorization decisions to improve performance and simplify the overall security management of the ABAC solution [144, 95]. Therefore, we consider the case of a hospital where the administrators have defined 6 attributes: *role*, *department*, and *speciality* for subjects, *type* and *owner* for objects, and *working hours* for the environment. It must be pointed out that different combinations of these attributes and their values are used to define access rules, and that a rule does not necessarily include all the attributes. Moreover, we tested the run-time performance of our model with up to 50 synthetic rules.

The execution time in function of the number of rules is shown in *Figure 4.4*. It is evident that the time will increase with the number of rules to be verified. However, the time difference does not follow a specific function because the number of attributes to be fetched is different from one rule to another. With 95% of confidence level we obtain a confidence interval of [29.138773, 34.305027].

In contrast, we found it interesting to see if the way in which the searching process is done, and the queries are executed, affects the performance. Therefore, we carried out the tests in two cases. The first case consists in gathering the timestamps when the attributes had the values defined in the security policy and then evaluating the rules according to EC, as discussed earlier. The second case is about gathering the values that the attributes had at the time of the access, and then evaluating the rules.
We present the evolution of the execution time in function of the number of attributes to be validated in both cases in *Figure 4.5*.

The difference between the two cases are the queries that are executed by the agents to get the attributes values. For instance, instead of answering the question "at

Figure 4.4: Time in function of number of rules

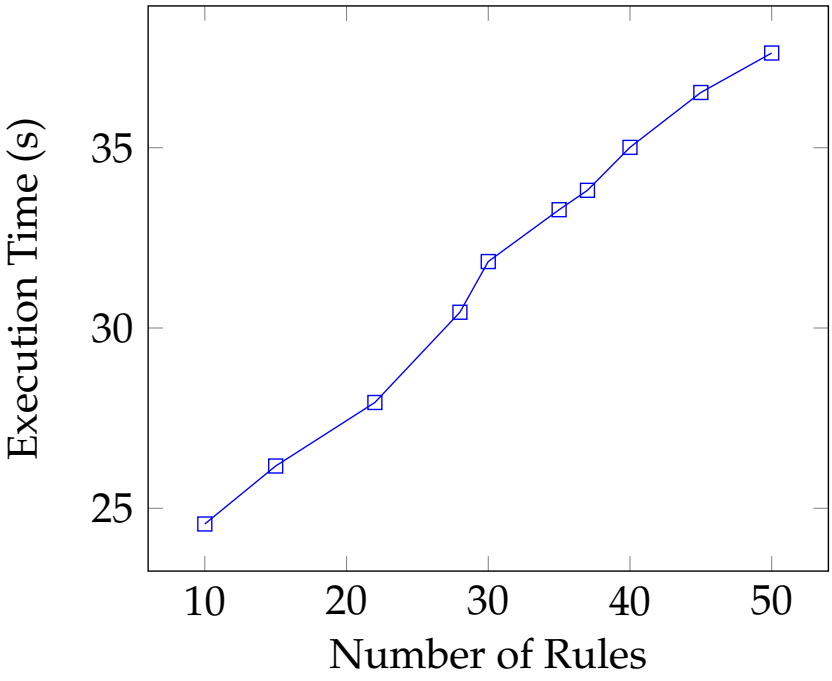
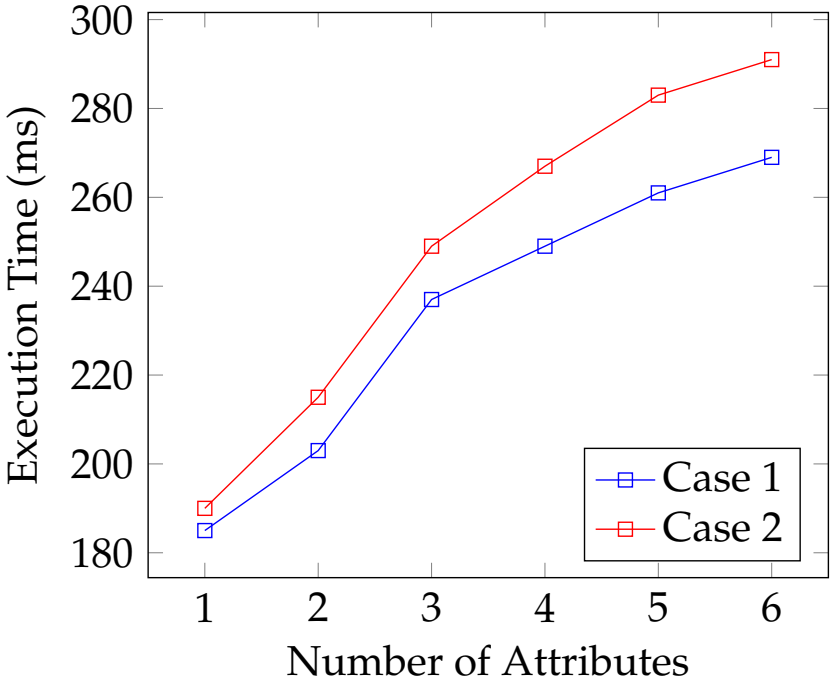


Figure 4.5: Time in function of number of attributes



what time did the user *x* have the role doctor?", the agents will answer the question "which role did the user have at access time?". Since we considered that the data sources contain the events of assigning/removing an attribute, and their timestamps, the search in the second case considers getting all the events that affect a certain attribute that occurred before the logged timestamp (it is hardly realistic to have the attribute values logged all the time). For instance, ***Med*** will ask $\boldsymbol{DS_1}$ which roles were set/removed to the user *9000000003* (*setRole(9000000003 ,?x)*) before *2019-07-22 14:59:04*. Therefore, the queries executed by $\boldsymbol{DS_1}$ are more complex than before, and result in a greater number of events. For example, other attribute's values can be included in the answer, such as the roles *Nurse* or *Lab Technician*, and not only *Doctor*. That being said, more individuals in the EC ontology are generated in this case; hence, longer reasoning, which justifies the results obtained in *Figure 4.5*.
In consequence, it is important to specify in the searching process not only the type of the attribute defined in the rule, but also its required value, to have a better performance.

4.7 Related Work

A large body of research on usage control was grounded on the idea of changes that can be done on subject and object attributes over time. For example, [160] defined a taxonomy for attribute management to show how attributes can be controlled in usage control. However, while they show how usage control can apply this mutability property of attributes in various traditional access control policies, we perform an a posteriori access control by verifying the attributes values, at the time of the logged event, to take violations decisions. Thus, their problematic is different from ours as the context in which attributes mutability is treated stands out.

Since a good log analysis leads to good decisions for accountability, it is fundamental to enrich the logs with complementary information related to the security policy, while incorporating a mechanism for temporal conditions verification. We thus study topics that are related to temporal access control.

Extending traditional access control models was a main interest of many researches.

For instance, in [24], a temporal extension of the role based access control model (TRBAC) was presented. The main features of this extension were the support for periodic enabling/disabling of roles, individual exceptions, and the possibility of specifying temporal dependencies among such actions, expressed by means of role triggers. TRBAC was then improved in [110] to be more generalized, and capable of expressing a wider range of temporal constraints such as duration constraints on roles, user-role, and role-permission assignments.

On the other hand, the Event Calculus has been proved to be powerful when it comes to access control security policies. In this respect, [17] showed how security models concerning the Discretionary Access Control can be represented using the Simplified Event Calculus (SEC). Yet, their work enforces the a priori access control, and the use of DAC is not expressive enough. Besides, [175] described the use of Event Calculus for developing a language that supports specification and analysis of authorization policies for Web service composition. Moreover, in [66], the authors used Event Calculus and abductive reasoning to develop an expressive language to analyze policy-based systems. The language combines authorization, obligation and refrain policies, and the abductive analysis is used to detect modality conflicts and a range of application-specific conflicts. However, because we are working on the a posteriori access control, our approach is based on the deductive Event Calculus, as the history of the logged events is used to derive the attributes values (the fluents) that held at the time of the event. In addition, [203] showed how a range of temporal RBAC (TRBAC) security models can be represented as logic programs incorporating the Simplified Event Calculus (SEC), that valorizes time-constrained permissions and roles membership. It also showed how clausal form logic expressing integrity constraints can enforce high-level security requirements.

4.8 Conclusion

In this chapter, we proposed a model to detect policy violations, in the a posteriori access control, based on a multi-agent system and the Event Calculus. We showed how the multi-agent system can be very helpful in collecting the necessary policy-related information to complement logged data. It eliminates human tasks by automating the

collection process.

Moreover, we integrated the problem of policy temporal compliance with the semantic enrichment process to highlight its importance when performing an a posteriori compliance check. This added value of temporal verification was achieved using the Event Calculus, that we modelled in SWRL. We demonstrated that the Event Calculus is expressive enough to model this kind of problem as it has direct support for representing logged events as well as the security policy. In addition, its formalism assures the correlation between these two essential components to detect violations. We also used the deductive reasoning supported by the Event Calculus, as the history of events occurring in the system is used to derive the fluents that will hold.

Nevertheless, in reality, like security attributes, security rules are also subject to change over time. As we presented the case of a static security policy in this chapter, we found it interesting to treat the case of the a posteriori access control, where logs are governed by an evolutive security policy that changes using administrative actions. We thus, present this case in the following chapter.

Chapter 5

A Posteriori Violation Detection with an Evolutive Policy

5.1 Introduction

For security concerns, organizations need to analyze and review their logs from one time to another to make sure that their deployed security policy is being respected. As discussed earlier, this process is associated with the a posteriori access control. We have presented, in the previous chapters, solutions to handle the first two steps of this type of access control, that are logging and auditing. Nevertheless, to accomplish the monitoring process, our violation detection mechanism relied on the semantic enrichment of logs. In addition, we highlighted a vital aspect that is introduced by the fact of having *a posteriori* investigations, that is *time*. It is thus, fundamental to take into account the temporal compliance when analyzing logs as many changes can take place between the time of the access and the time of the investigation. We focused mainly, on the variation of the security attributes and their values over the time, and we considered that the security policy is static.

However, changes can be applied on security rules too; hence, the policy itself can evolve over time to include more or less rules. These types of modification can be done using an administrative security policy.

Indeed, to have a complete access control model, an administration model should be provided. For instance, the RBAC model was associated with ARBAC97 [178], the

OrBAC model was affiliated with AdOrBAC [52], and the ABAC model was recently paired with AMABAC [106]. These models control who is permitted to assign/revoke attributes and/or permissions; hence, modifying rules in the security policy.

Another thing to be pointed out is that administrators themselves should have the right privileges in order to modify the security policy. In consequence, administrative actions should also be monitored to check that the applied modifications were also allowed.

All previous works related to the a posteriori access control considered a static security policy and did not take into account its time dependent evolution. In this chapter, we re-solicit the Event Calculus to include the temporal evolution of the security policy in our violation detection mechanism. We thus, consider changes that can affect the security rules using an administrative policy model, as well as the violations that can be caused by both the users and the administrators. Moreover, we formalize the violation detections as a recursive process and show its termination.

5.2 Motivation of Considering Policy Evolution

Many attempts have been made to adapt to changes in traditional access control [79, 50, 214]. For instance, [214] presented the Dynamic Role Based Access Control (DRBAC) model that provides context-aware access control for pervasive applications. DRBAC extends RBAC and dynamically adjusts role assignments and permission assignments based on context information. However, it has been shown that it must be combined with authentication mechanisms to secure pervasive applications in real life. Moreover, the authors in [122], introduced the notion of consistency of access rules in a collaborative environment, and addressed the problem of maintaining consistency through occasional changes. As they treated the a priori access control, and policy changes may occur while queries are actively being processed, these changes were accommodated online to synchronize and modify query planning.

In the area of the a posteriori access control, previous works that we presented in *Section 2.1.2* such as [67, 37, 77] assumed that the policy is correct and static when evaluating its compliance. Besides, some works found that policy reconciliation

can be done by reconstructing the policy from logs. For example, in [40] , the authors proposed an approach to verify the enforcement of security policies and the usage of permissions. Their method was based on analytics, and attempts to ensure the consistency of the used permissions with the configured policy, in addition to guaranteeing that the policy maintains least privilege using unambiguous constructs to reduce administrative errors. The consistency was provided by mining roles from usage logs and checking their correspondence with the actual policy. [123] identified anomalies in RBAC models that may indicate insider threats by comparing a prescriptive RBAC model to a generative RBAC model that can be derived from event logs. Furthermore, they provided metrics for structural and semantic differences between RBAC models, and used visualization techniques for evaluation.
In both of these works usage mining was used to compare logs to the security policy. However, roles, for example, are mined as general behaviour, making it impossible to distinguish which role was used by a user at a specific time point.

As discussed above, related works on reconciling policy considered a static security policies. In order to take right decisions for accountability purposes, security auditors need to have a correct reference. Therefore, it is important to check which rules were in place when an access was done, and to monitor administrators' actions as they can also be accountable. Our literature review has showed that this problem was not treated before.

In contrast, to confirm the importance of considering the evolution of the policy when performing an a posteriori access control, we provide below a motivating example:
The *"Stay Alive"* hospital has deployed a "break-glass" mechanism, where access authorizations outside the standard case can be given explicitly, on a case-by-case basis, by the administrator. *Mary* and *Jeanne* are two nurses who work, respectively, in the cardiology and the neurology departments of the hospital.
At the beginning of spring, *Mary* took a vacation for two weeks. Since nurses are only allowed to access medical records that belong to the same department in which they work, *Jeanne*, who replaced *Mary* during that period of time, asked the administrator to grant her the necessary accesses to complete the job. During that same period, *Jeanne* has viewed the medical record of a patient in the oncology department. Did *Jeanne*

have the right to do so? Was the created rule correct? Did the administrator abuse his privileges? Etc.
The goal of this chapter is to answer these questions.

5.3 Administrative Models for ABAC

In *Chapter 4*, we modelled the security policy according to ABAC for its support of making fine-grained access decisions, and capability of supporting any access control model. We thus, recall that in ABAC, access is granted according to user attributes, resource attributes, action attributes, and context attributes.

As any access control model, ABAC needs to have an administrative representation. In the following, we briefly present the administrative models for ABAC that were proposed in the literature, and we justify our choice of using AMABAC, which we present in details.

5.3.1 GURA

The first effort for developing an administrative model for ABAC was [107], where a Generalised User-Role Assignment (GURA) was proposed, and which consists of a set of administrative requests and a set of administrative rules. In GURA, user attributes are collectively administered by different administrative roles to enable distributed administration. Therefore, in this administrative model, the administrative policy specifies the conditions under which administrative roles can modify user attributes through administrative requests. These requests take effect only if they are authorized by administrative rules. However, GURA relies on the set of relations defined in ARBAC97 [178]. Moreover, it is likely best suited to user attributes which makes him inappropriate for administrating other attributes such as objects and environmental attributes.

5.3.2 ADABAC

Another proposed administrative model for ABAC was the Administrative ABAC (ADABAC) [105]. ADABAC supports decentralized administration of ABAC systems, and consists of a number of operations to administer the set of subjects and the set of subject attribute assignments in an ABAC system. Moreover, an administrative rule in ADABAC essentially associates a set of administrative users having certain attribute name-value pairs to a set of administrative operations. Each operation in ADABAC has one or more preconditions that need to be satisfied prior to the execution of the operation, and certain postconditions should also hold after the execution of the operation. Therefore, ADABAC is similar to GURA by the fact that it can only be used to manage subject-related components and does not include any components for managing object and environmental related components or policies. Therefore, we chose the ABAC administrative model AMABAC [106] since it provides solutions for these problematics.

5.3.3 AMABAC

AMABAC is an Administrative Model for ABAC, where a set of authorized administrative users U, who have a set of administrative attributes A, that can acquire possible values R_a, and a set of administrative relations AP, are defined. The set of attribute name–value pairs associated with an administrative user a is given by the expression *attr(a)*.
Each administrative relation $Re_i \in AP$ is of the form <*ac*,*Par*>, where *ac* is an administrative attribute condition, that is a set of administrative attribute name-value pairs, and *Par* is an optional set of parameters passed to the relation Re_i. The role of these relations is to define the set of attributes that an administrative user must have to be able to modify a specific component in an ABAC system.

Furthermore, there are 20 administrative relations and commands in AMABAC, that are meant to modify subject, object, and environmental attribute-related components, as well as authorization rules-related components. In this chapter, we are interested in the rules' modifications as a whole.

Two administrative relations were defined for authorization rules modification in AMABAC. The first relation is to add a rule, and the second one is to remove a rule from the policy.
Moreover, each relation is associated with an administrative command that is required to be executed to perform the actual modification. In this connection, some preconditions need to be satisfied prior to the execution of a command for this latter to take effect on the policy. For instance, a new rule r can be added (removed) to the policy P if there is an administrative user a who has the administrative attribute ac ($ac \subseteq attr(a)$) that allows the insertion (removal) of a new rule (*can_add_rule(ac)*), and if the same rule r is not already in (is already in) the policy P ($r \in P$).

Yet, the fact of verifying if the rule belongs to the security policy or not before adding it is not enough. The verification should go further than that to include redundancy checking [86], as well as conflict resolution [143, 187]; hence, the lack of these types of verification represent a limitation in AMABAC.
Therefore, to simplify the problem, we will not take into account this requirement as a necessary precondition, and will not consider the verification of the preconditions $r \notin P$ and $r \in P$, when respectively adding and removing a rule.

It is worth to mention that we model the AMABAC policy in OWL.
As AMABAC is also attribute-based, the ontology representing it is similar to the ABAC's ontology presented in *Section 4.3*, where *administrative users, administrative actions, rules, permissions,* and the *policy* are represented as Classes, and their predicates *subject*, *object*, etc., are represented as properties.

5.4 Evolutive Policy Compliance

Once again, we consider that a logged event is retrieved using the semantic mediator presented in *Chapter 3*. This event is of the form $e = (u, op, o)$, where op is an action done by a user u on an object o.

Conversely, an ABAC rule presumes that an action is permitted if the subject, object, and environment that are involved in it fulfill certain values. In consequence, we denote a security rule as $r = <SA, OA, EA, op>$, where SA, OA, EA are the required

attributes' values of the subject, object, and environment, respectively, and *op* is a permitted action. Moreover, ABAC rules contain an *if-then* statement, in which the satisfiability of a condition (*the required attributes' values*) leads to a permitted action; hence, a security rule can be expressed as *Condition* $\longrightarrow$ *is-permitted(u,op,o)* (*c.f. Section 4.4.2)*.

In the previous chapter, we developed a framework for policy violation detection in case of static security rules, that remain the same over time. We have shown how the expression defining *Always* can be applied to each rule as in expression (4.3). Therefore, since the rules are always true, we only had to check if *SA, OA,* and *EA* were satisfied at the time the action was executed.

Nevertheless, we consider now the case of an administrative security policy, where an administrator can change the deployed security rules over time. The analysis consists then not only in checking if the condition held at the time of the access, but also in verifying which rule that relates the condition to the permitted action was valid at that time. Therefore, *Always* cannot be applied anymore.

We suppose that at the time of the investigation t_{invest} (now), the policy is in its last updated state, and that the logged event to be analyzed happened at $t < t_{invest}$. Thus, we distinguish two types of verification: (1) check if the logged action at the past time t was permitted or not, by fetching the rules that were in the policy at that same time t, and collect the user, object, and context attributes defined in the holding rules for verification, and (2) check if the rules corresponding to the deployed policy at time t were created by administrators who had the right to create them.
That being said, we consider having two log databases: one that registers all the actions executed in the application domain by regular users, and one that records administrative actions. It must be pointed out that the separation between the two databases is purely conceptual. We could consider that there is only one database, but we made that choice to distinguish between regular and administrative actions.

Moreover, the deployed multi-agent system architecture remains the same. However, in this chapter, the ***Policy Agent (Po)*** will have access to the administrative log in which the modifications of security rules that are done by administrators are traced. The rest of the agents have the same functionality and the message exchange

process is identical. In the following, we mainly focus on the formalization of the problem in the Event Calculus rather than the functioning of the multi-agent system. Nevertheless, it is easy to sketch the messages exchanged between the different agents by referring to *Chapter 4*.

Before explaining how the violation detection can be done in case of an evolutive security policy, we introduce two new axioms in the Event Calculus. Since security rules are not changed frequently in reality, we provide interval manipulation to express succinctly the duration or period of time for which a security rule holds. Therefore, a fluent f holds for an open-closed interval I = $]t_1,t_2]$ as follows:

$$\begin{aligned} &Happens(e_1,t_1) \wedge Initiates(e_1,f,t_1) \wedge Happens(e_2,t_2) \wedge \\ &Terminates(e_2,f,t_2) \wedge \neg Clipped(t_1,f,t_2) \\ &\rightarrow HoldsFor(f,t_1,t_2) \end{aligned} \tag{5.1}$$

Always with respect to the reification problem, axiom (5.1) can be expressed in SWRL as follows:

```
Happens(?happens1) ∧ hasEvent(?happens1,?e1) ∧
hasTime(?happens1,?t1) ∧ Initiates(?initiates) ∧
hasEvent(?initiates,?e1) ∧ hasFluent(?initiates,?f)
∧ hasTime(?initiates,?t1) ∧ Happens(?happens2) ∧
hasEvent(?happens2,?e2) ∧ hasTime(?happens2,?t2) ∧
Terminates(?terminates) ∧ hasEvent(?terminates,?e2) ∧
hasFluent(?terminates,?f) ∧ hasTime(?terminates,?t2) ∧
NotClipped(?notClipped) ∧ hasFluent(?notClippend,?f) ∧
hasStartTime(?notClipped,?t1) ∧ hasEndTime(?notClipped,?t2)
∧ swrlx:makeOWLThing(?holdsFor,?f) → HoldsFor(?holdsFor)
∧ hasFluent(?holdsFor,?f) ∧ hasStartTime(?holdsFor,?t1) ∧
hasEndTime(?holdsFor,?t2)
```

It must be noted that the interval is open-closed since an event has an effect on a fluent right after its occurrence. For instance, when an initiating event happens at *t*, the corresponding fluent will start holding right after *t* (at *t+1)*. Similarly for the terminating event.

In particular, a fluent can be started by an event and not terminated yet. In this case, the fluent will hold until *now*:

$$\begin{aligned} &Happens(e, t_1) \wedge Initiates(e, f, t_1) \wedge \neg\exists t_2[t_1 < t_2 \wedge \\ &Clipped(t_1, f, t_2)] \rightarrow HoldsFor(f, t_1, now) \end{aligned} \tag{5.2}$$

Now that we have introduced these axioms, we start the policy evaluation process.

5.4.1 Getting Access Time Valid Rules

Considering the administrative security policy according to the AMABAC model, a permitted action will hold at the time of its execution t, if the *postconditions* triggered by an administrative command hold at that time t. In consequence, an action done by a user on an object is permitted if there is a rule that assures its permission at the time of its execution as follows:.

$$\begin{aligned} &Happens(e, t) \wedge e = (u, op, o) \wedge \exists r[HoldsFor(r \in P, t_1, t_2) \\ &\wedge t_1 < t <= t_2 \wedge HoldsAt(matches(A(e), A(r)), t)] \\ &\longrightarrow HoldsAt(is\text{-}permitted(u, op, o), t) \end{aligned} \tag{5.3}$$

We recall that r = <*SA, OA, EA, op*>, and t_1 and t_2 are the times when the rule r was added and removed from P, respectively. We also define accordingly *A(e)* and *A(r)* as the set of attributes concerned in the event and defined in the rule, including operations. The *matches* predicate returns true if the attributes' values and operation in *A(e)* are the same as the ones defined in *A(r)*, and false otherwise. The expression of the *matches* predicate in SWRL is given as follows:

```
SuperFluent(?f) ∧ isRelatedTo(?f,?e) ∧ isDefinedIn(?f,?r) ∧
swrlx:makeOWLThing(?m,?e) → matches(?m) ∧ hasDomain(?m,?e) ∧
hasRange(?m,?r)
```

As discussed in *Section 4.4.2*, the *SuperFluent* constitutes the condition fluent that includes all the attributes defined in a rule and that is related to the investigated event. It is also worth mentioning that if a rule was added and never removed before t_{invest}

(now), then the fluent $r \in P$ will hold from t_1 to t_{invest} like in (5.2).

In consequence, expressing (5.3) in SWRL leads to the following:

```
Happens(?happens) ∧ hasEvent(?happens,?e) ∧
hasTime(?happens,?t) ∧ type(?e,?op) ∧ subject(?e,?u) ∧
object(?e,?o) ∧ context(?e,?c) ∧ HoldsFor(?holdsFor) ∧
ruleisInpolicy(?i) ∧ hasRule(?i,?r) ∧ hasPolicy(?i,?p)
∧ HoldsFor(?holdsFor) ∧ hasFluent(?holdsFor,?i) ∧
hasStartTime(?holdsFor,?t1) ∧ hasEndtine(?holdsFor,?t2)
∧ swrlb:lessThan(?t1,?t) ∧ swrlb:lessThanOrEqual(?t,?t2)
∧ HoldsAt(?holdsAt) ∧ matches(?m) ∧ hasDomain(?m,?e)
∧ hasRange(?m,?r) ∧ hasFluent(?holdsAt,?m) ∧
hasTime(?holdsAt,?t) ∧ swrlx:makeOWLThing(?holdsAt,?happens)
→ HoldsAt(?holdsAt) ∧ isPermitted(?e) ∧ hasFluent(?holdsAt,?e)
∧ hasTime(?holdsAt,?t)
```

According to axiom (5.1), *"a rule r is in the security policy"* holds for an interval $]t_1,t_2]$, if it was added at t_1, removed at t_2, and not removed in the meantime. Thus, from the administrative log, we should search for the activating and deactivating events of the fluent $r \in P$, that are *add_rule and remove_rule*, respectively, if any. However, as mentioned earlier, in AMABAC, a set of preconditions should be satisfied for a successful execution of an administrative command. Therefore, we consider that when an administrator *a* has an administrative attribute which allows him to perform a certain action (*e.g., can_add_rule(ac) & ac* $\subseteq$ *attr(a)*), the action provided by that attribute becomes permitted (*e.g., is-permitted(a,add_rule,r)*). Thus, a rule is successfully added to the policy as follows:

$$\begin{aligned} &HoldsFor(\textit{is-permitted}(a, add\_rule, r), t_1, t_2) \wedge \\ &Happens((a, add\_rule, r), t) \wedge t_1 < t <= t_2 \\ &\longrightarrow Initiates((a, add\_rule, r), r \in P, t) \end{aligned} \tag{5.4}$$

The expression of (5.4) in SWRL is given below:

```
HoldsFor(?holdsFor) ∧ isPermitted(?e) ∧ subject(?e,?a) ∧
type(?e,add_rule) ∧ object(?e,?r) ∧ hasFluent(?holdsFor,?e)
∧ hasStartTime(?holdsFor,?t1) ∧ hasEndTime(?holdsFor,?t2)
∧ Happens(?happens) ∧ hasEvent(?happens,?e) ∧
hasTime(?happens,?t) ∧ swrlb:lessThan(?t1,?t) ∧
swrlb:lessThanOrEqual(?t,?t2) ∧ swrlx:makeOWLThing(?initiates,
?happens) ∧ swrlx:makeOWLThing(?i,?happens) →
Initiates(?initiates) ∧ hasEvent(?initiates,?e) ∧
hasFluent(?initiates,?i) ∧ ruleisInpolicy(?i) ∧ hasRule(?i,?r)
∧ hasPolicy(?i,p) ∧ hasTime(?initiates,?t)
```

Similarly, we can express the case of removing a rule from the policy:

$$\begin{aligned} &HoldsFor(is\text{-}permitted(a, remove\_rule, r), t_1, t_2) \wedge \\ &Happens((a, remove\_rule, r), t) \wedge t_1 < t <= t_2 \\ &\longrightarrow Terminates((a, remove\_rule, r), r \in P, t) \end{aligned} \tag{5.5}$$

The expression of (5.5) in SWRL is:

```
HoldsFor(?holdsFor) ∧ isPermitted(?e) ∧ subject(?e,?a) ∧
type(?e,remove_rule) ∧ object(?e,?r) ∧ hasFluent(?holdsFor,?e)
∧ hasStartTime(?holdsFor,?t1) ∧ hasEndTime(?holdsFor,?t2)
∧ Happens(?happens) ∧ hasEvent(?happens,?e) ∧
hasTime(?happens,?t) ∧ swrlb:lessThan(?t1,?t) ∧
swrlb:lessThanOrEqual(?t,?t2) ∧ swrlx:makeOWLThing(?terminates,
?happens) ∧ swrlx:makeOWLThing(?i,?happens) →
Terminates(?terminates) ∧ hasEvent(?terminates,?e)
∧ hasFluent(?terminates,?i) ∧ ruleisInpolicy(?i) ∧
hasRule(?i,?r) ∧ hasPolicy(?i,p) ∧ hasTime(?terminates,?t)
```

5.4.2 Monitoring Administrative Actions

As stated in (5.4) and (5.5), the effect of an administrative action on the security policy depends on the administrative attributes and rights that the administrator who is performing the action has. Thus, we still need to check if that administrator has the right to perform a modification action (*e.g., HoldsAt(is-permitted(a,add_rule,r), t)*). In consequence, we need to apply (5.1) again for the fluents *is-permitted(a,add_rule,r)* and *is-permitted(a,remove_rule,r)*, by getting the appropriate initiating and terminating events from the administrative log. In the same way, we need to verify if the administrative user who is modifying the administrators' attributes also has the right attributes to do so. Therefore, a recursive process is introduced as shown in *Figure 5.1*.

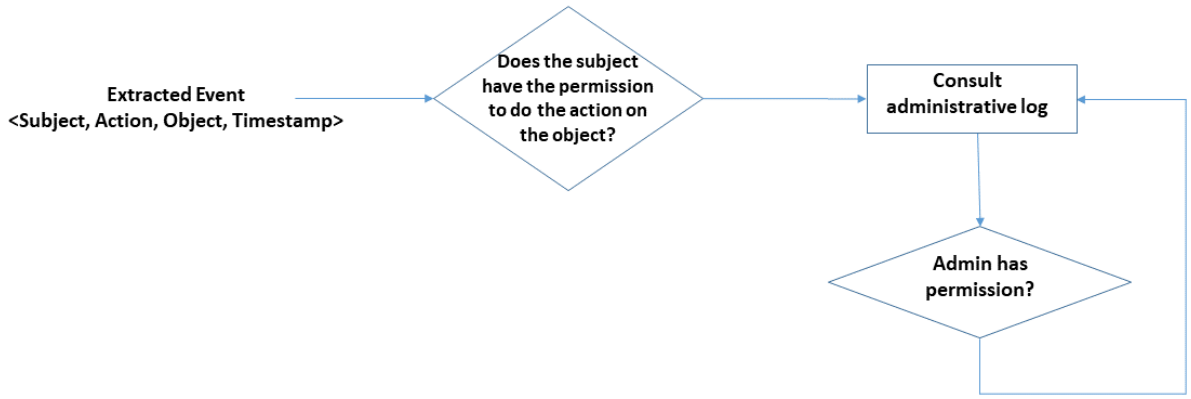


Figure 5.1: Recursive Aspect of administrative attributes verification.

To put an end to this verification loop, we consider that there is only one administrator, whom we call *super administrator sad*, who can assign/remove administrative rights to the rest of the administrators. This proposition will define the initial states, since at the time of the conception of the application only one administrator will delegate permissions to other administrators; hence, this will guarantee the end of the recursion. Consequently, we extend AMABAC to include two new administrative commands, *assign_admin_perm*, and *remove_admin_perm*, that can be executed by *sad* without any precondition. Therefore, we obtain the following:

$$Always(is\text{-}permitted(sad, assign\_admin\_perm, perm)). \tag{5.6}$$

Similarly, all the other administrative actions such as *remove_admin_perm*, *add_rule*, *remove_rule*, etc., are always permitted for *sad*.

Moreover, we define a permission *perm* as a tuple <*owner, action, condition*>, where *owner* is the owner (administrator) to whom the permission is assigned, *action* is the operation that is allowed by the permission, and *condition* is the condition that should be satisfied by the object on which the permission is applicable. For instance, *perm*=<a_1, *add_rule, rule.SA=(role=doctor)*> is a permission where the administrator a_1 can add rules in which the subject has the role doctor. It is worth mentioning that as in AMABAC the administrative actions are general (e.g., *add_rule* permits creating any rule), we added the *condition* element to have more expressivity and preciseness in the actions that an administrator can do.

As a result, the loop stops once it gets to the super administrator who has, for sure, the right to perform any action. This idea is depicted in *Figure 5.2*.

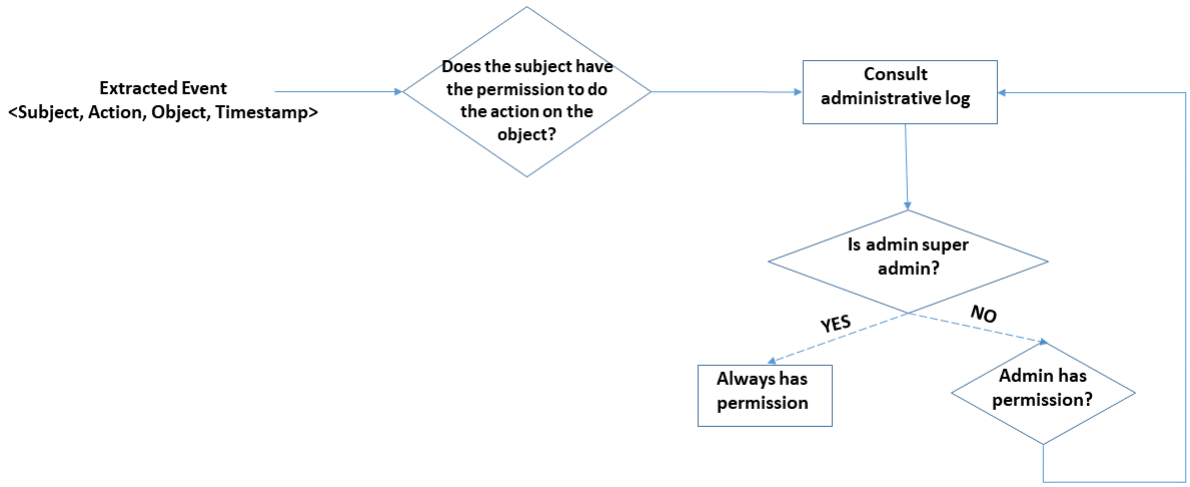


Figure 5.2: Recursive Aspect ends when *sad* is detected.

Now that *sad* is assigning/removing the administrative permissions of other administrative users, the corresponding actions will be permitted for the defined user without any dependency of other actions, as follows:

$$\begin{aligned} &Happens((sad, assign\_admin\_perm, perm), t) \wedge \\ &\exists o[satisfies(o, perm.condition)] \longrightarrow Initiates((sad, \\ &assign\_admin\_perm, perm), is\text{-}permitted(perm.owner, \\ &perm.action, o), t) \end{aligned} \tag{5.7}$$

The expression of (5.7) in SWRL is:

```
Happens(?happens) ∧ type(?e,assign_admin_perm) ∧
subject(?e,sad) ∧ object(?e,?perm) ∧ hasEvent(?happens,?e)
∧ hasTime(?happens,?t) ∧ condition(?perm,?cond) ∧
owner(?perm,?ow) ∧ action(?perm,?act) ∧ Object(?o) ∧
satisfies(?o,?cond) ∧ swrlx:makeOWLThing(?initiates,?happens)
∧ swrlx:makeOWLThing(?e2,?happens) → Initiates(?initiates) ∧
hasEvent(?initiates,?e) ∧ type(?e2,?act) ∧ subject(?e2,?ow) ∧
object(?e2,?o) ∧ isPermitted(?e2) ∧ hasFluent(?initiates,?e2)
∧ hasTime(?initiates,?t)
```

$$\begin{aligned} &Happens((sad, remove\_admin\_perm, perm), t) \wedge \\ &\exists o[satisfies(o, perm.condition)] \longrightarrow Terminates((sad, \\ &remove\_admin\_perm, perm), is\text{-}permitted(perm.owner, \\ &perm.action, o), t) \end{aligned} \tag{5.8}$$

The expression of (5.8) in SWRL is:

```
Happens(?happens) ∧ type(?e,remove_admin_perm) ∧
subject(?e,sad) ∧ object(?e,?perm) ∧ hasEvent(?happens,?e)
∧ hasTime(?happens,?t) ∧ condition(?perm,?cond) ∧
owner(?perm,?ow) ∧ action(?perm,?act) ∧ Object(?o) ∧
satisfies(?o,?cond) ∧ swrlx:makeOWLThing(?terminates,?happens)
∧ swrlx:makeOWLThing(?e2,?happens) → Terminates(?terminates) ∧
hasEvent(?terminates,?e) ∧ type(?e2,?act) ∧ subject(?e2,?ow) ∧
object(?e2,?o) ∧ isPermitted(?e2) ∧ hasFluent(?terminates,?e2)
∧ hasTime(?initiates,?t)
```

5.4.3 Detecting violations

Once all the necessary administrative events are obtained and after resolving expressions (5.4) - (5.8), we can get the rules that were in the policy at the time when the log event, to be checked, was executed. In this respect, we obtain from these rules all the required attributes that should be satisfied by the user and object at that same time; hence, we can start searching for these attributes in different databases to contextualize the extracted event and verify if they are compliant with the ones defined in the policy rules (*HoldsAt(matches(A(e),A(r)),t)*) as in (5.3). This step goes back to *Chapter 4*, where the ***Data Source (DS)*** agents search for the user, object, and environmental attributes, by consulting the administrative log (the history log) of each data source to see when an administrator assigned/removed an attribute to a regular user. Thus, the other administrative relations, and commands of AMABAC, that permits modifying subject, object, and environmental attribute-related components, are solicited. Additionally, the verification of administrative privileges is also required in this step. An example of the interaction between the validity of security rules and administrative actions is illustrated in *Figure 5.3*.

Finally, if the executed action was not permitted at the time of the access, we can deduce a violation like in expression (4.7).

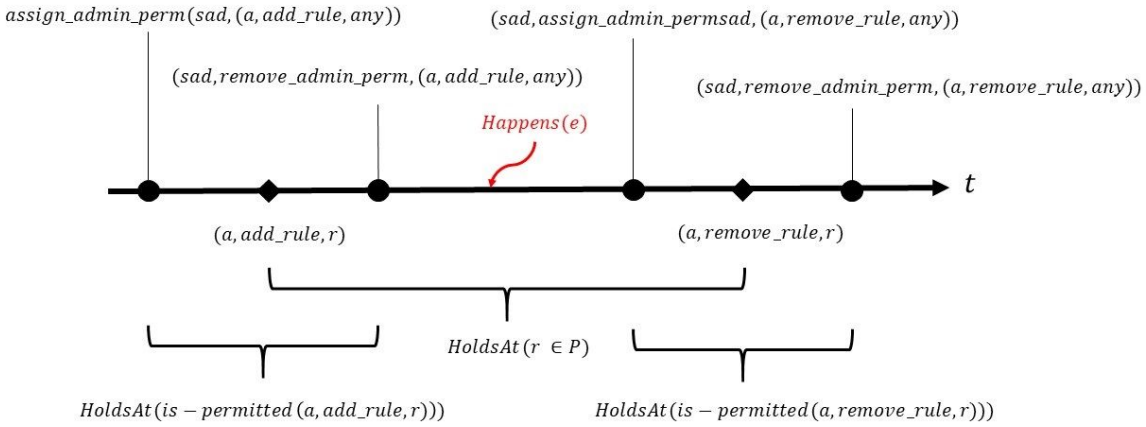


Figure 5.3: Timeline example of interactions between security rules and administrative actions.

5.5 Use Case

We illustrate the violation detection taking into account the evolution of security rules with a practical example.

Using the same EHR application, we consider two administrators a_1 and a_2, and one super administrator *sad*. At the time of the creation of the application (t=0), *sad* assigned the privileges of adding and removing rules in which the subject is a doctor to a_1, and adding and removing rules concerning all the users except doctors to a_2. *Figure 5.4* shows an excerpt of the corresponding administrative log.

sad	assign_admin_perm	add_rule	a1	r.SA = (role = doctor)	t=0
sad	assign_admin_perm	add_rule	a2	r.SA = (role != doctor)	t=0
sad	assign_admin_perm	remove_rule	a1	r.SA = (role = doctor)	t=0
sad	assign_admin_perm	remove_rule	a2	r.SA = (role != doctor)	t=0
		.			
		.			
		.			
a2	add_rule	r1			t=16
a1	add_rule	r2			t=18
a1	add_rule	r3			t=24
a2	remove_rule	r1			t=33
a2	remove_rule	r3			t=40

Figure 5.4: Excerpt of the administrative log.

We consider three ABAC rules, that were added and/or removed by a_1 and a_2 at different times as follows:

r_1:*"A lab technician can create a lab procedure".*

r_2:*"A doctor can create a prescription during an office visit".*

r_3:*"A nurse can view a medical record that is in the same department in which he/she works in".*

Furthermore, we suppose that we extracted from the application log the following event, which we want to check if it is a violation or not:

e = (9000000003, CREATE, PRE35876, 35), and we consider having one policy named p_1.

Besides, the investigation is done at *t = 45* (now). We also define the conditions c_i and permissions $perm_i$ as follows:

c_1=<*r.SA=(role=doctor)*>

c_2=<*r.SA=(role!=doctor)*>

$perm_1$=<a_1,*add_rule*,c_1>

$perm_2$=<a_2,*add_rule*,c_2>

$perm_3$=<a_1,*remove_rule*,c_1>

$perm_4$=<a_2,*remove_rule*,c_2>

By expressing the administrative log events in the Event Calculus, we obtain the following:

Happens((sad, assign_admin_perm, perm$_1$), 0)

Happens((sad, assign_admin_perm, perm$_2$), 0)

Happens((sad, assign_admin_perm, perm$_3$), 0)

Happens((sad, assign_admin_perm, perm$_4$), 0)

Happens((a_2, add_rule, r_1), 16)

Happens((a_1, add_rule, r_2), 18)

Happens((a_1, add_rule, r_3), 24)

Happens((a_2, remove_rule, r_1), 33)

Happens((a_2, remove_rule, r_3), 40)

Next, by applying (5.7) and (5.8), we get:

Initiates((sad,assign_admin_perm,perm$_1$),is-permitted(a_1, add_rule, r_2), 0)
Initiates((sad,assign_admin_perm,perm$_3$),is-permitted(a_1,remove_rule,r_2),0)
Initiates((sad,assign_admin_perm,perm$_2$),is-permitted(a_2,add_rule,r_1),0)
Initiates((sad,assign_admin_perm,perm$_2$),is-permitted(a_2,add_rule,r_3),0)
Initiates((sad,assign_admin_perm,perm$_4$),is-permitted(a_2,remove_rule,r_1),0)
Initiates((sad,assign_admin_perm,perm$_4$),is-permitted(a_2,remove_rule,r_3),0)

Since the fluents *is-permitted(a_i,add_rule,r_j)* and *is-permitted(a_i,remove_rule,r_j)* were never terminated, they hold until now:

HoldsFor(is-permitted(a_i, add_rule, r_j), 0, now)
HoldsFor(is-permitted(a_i, remove_rule, r_j), 0, now).

In addition, (5.4), and (5.5) lead to having the following:

HoldsFor($r_1 \in p_1$, 16, 33)
HoldsFor($r_2 \in p_1$, 18, now)

In consequence, the only rule that held at $t = 35$ was r_2.

Now that we know which rule was valid at the time of the access, we can start searching for the defined attributes (e.g., the subject's role, the subject's department, the type of object, the object's department, the context, etc.) to see if they verify expression (5.3), and whether there was a violation or not according to (4.7).
Supposedly that *9000000003* is the identifier of a doctor, and the object *PRE35876* has the type prescription, but the condition does not respect an office visit, then a violation of r_2 is induced (*c.f. (5.3) and (4.7)*). Therefore, the user *9000000003* should be held accountable for not respecting the security rule that was in place when he created the prescription.

Now, we assume that we want to investigate the event *e=(7000000005,VIEW,MR8853,37)*. The administrative log as well as the investigation time remain the same. Besides, we consider that all the requirements of r_3 are fulfilled

by the occurring event (*7000000005* is a nurse who viewed the medical record *MR8853* that is in the same department she works in). In this respect, when searching for the rules that were holding at $t = 37$, the answer will include only rule r_2, and two violations will be deduced. This can be explained with the following: according to the administration log, a_1 was assigned the permission to add rules concerning doctors only; hence, following (5.7), the fluent *is-permitted*(a_1, *add_rule*, r_3) was never initiated. Thus, a violation is produced according to (4.7). Continuing with (5.4), a rule is added successfully to the policy if the administrator who is performing the action has the right to do so. Since it is not the case, the fluent $r_3 \in p_1$ is not initiated. As a result, (5.1) leads to r_3 not being in the policy, which justifies why we will only get r_2 as holding rule at $t = 37$. Moreover, despite the fact that all the attributes are conform with the ones defined in r_3, and that the user technically did not violate the rule as it was added by a_1, a second violation will be provoked by resolving (5.3) and (4.7), since only r_2 was *legally* in place at the time of the access and the attributes defined in it are not respected.

At this point where violations are detected, responsibilities should be fixed to account the users and administrators. When only regular users violate the rules that were in place, the process is simpler because only them should justify their actions. Nevertheless, the accountability becomes more complex when the administrators themselves violate the administrative policy, the case in which not only administrators should prove the legitimacy of their operations, but also regular users. As the administrators should be sanctioned for sure, the decision concerning the users should be looked into. This is where collateral damage comes around, to fix the responsibility of users when their accesses were authorized by rules that were created by someone who had not the permission to do so. Therefore, the decisions can be made depending on the impacts and side effects of the actions on the system. We hereby treat this problem in the following chapter.

5.6 Implementation

The tools used to implement the approach proposed in this chapter are the same as the ones presented in *Section 4.6.1*. Moreover, it was mentioned that certain EC

predicates deal with causal constraints which makes it necessary to support existential quantification that is not supported in SWRL. Therefore, when dealing with EC, we always need to couple SWRL with an algorithm along with some SQWRL queries to have a correct implementation of the axioms. We recall that SQWRL queries can only work on known individuals (instances) in an ontology but they do not permit any alterations to the information that they might extract from the ontology.

In the following we present the SQWRL queries that we use in our algorithm.

StartRuleQuery: gets all the rules that were added to the policy by an *add_rule* event at a timepoint.

```
Initiates(?initiates) ∧ ruleisInpolicy(?i) ∧ hasRule(?i,?r)
∧ hasPolicy(?i,p1) ∧ hasFluent(?initiates,?f) ∧
hasEvent(?initiates,?e) ∧ type(?e,add_rule) ∧
hasTime(?initiates,?t) → sqwrl:select(?initiates,?i,?e,?r,?t)
```

The query returns the *Initiates* statements, together with their associated *Event, Fluent* and *timepoint* references.

EndRuleQuery: gets all the rules that were removed from the policy by a *remove_rule* event at a timepoint.

```
Terminates(?terminates) ∧ ruleisInpolicy(?i) ∧ hasRule(?i,?r)
∧ hasPolicy(?i,p1) ∧ hasFluent(?terminates,?f) ∧
hasEvent(?terminates,?e) ∧ type(?e,remove_rule) ∧
hasTime(?terminates,?t) → sqwrl:select(?terminates,?i,?e,?r,?t)
```

StartPermittedAdminActionQuery: gets the permitted administrative actions that were initiated by an *assign_admin_perm* event at a timepoint.

```
Initiates(?initiates) ∧ isPermitted(?e2) ∧
subject(?e2,?ow) ∧ object(?e2,?o) ∧ type(?e2,?act) ∧
hasFluent(?initiates,?e2) ∧ hasEvent(?initiates,?e) ∧
type(?e,assign_admin_perm) ∧ hasTime(?initiates,?t) →
sqwrl:select(?initiates,?e,?e2,?ow,?act,?o,?t)
```

EndPermittedAdminActionQuery: gets the permitted administrative actions that were terminated by a *remove_admin_perm* event at a timepoint.

```
Terminates(?terminates) ∧ isPermitted(?e2) ∧
subject(?e2,?ow) ∧ object(?e2,?o) ∧ type(?e2,?act) ∧
hasFluent(?initiates,?e2) ∧ hasEvent(?initiates,?e) ∧
type(?e,remove_admin_perm) ∧ hasTime(?terminates,?t) →
sqwrl:select(?terminates,?e,?e2,?ow,?act,?o,?t)
```

HoldsForQuery: gets the rules that were holding during an interval of time.

```
HoldsFor(?holdsFor) ∧ ruleisInpolicy(?i) ∧ hasRule(?i,?r)
∧ hasPolicy(?i,p1) ∧ hasFluent(?holdsFor,?i) ∧
hasStartTime(?holdsFor,?t1) ∧ hasEndTime(?holdsFor,?t2) →
sqwrl:select(?holdsFor,?i,?r,?t1,?t2)
```

HoldsAtPermittedUserEventQuery: gets the permitted user events that hold at a timepoint.

```
HoldsAt(?holdsAt) ∧ isPermitted(?e) ∧ subject(?e,?u) ∧
User(?u) ∧ hasFluent(?holdsAt,?e) ∧ hasTime(?holdsAt,?t) →
sqwrl:select(?holdsAt,?e,?u,?t)
```

ViolationQuery: gets the violations.

```
Violation(?e) → sqwrl:select(?e)
```

The pseudo-code of the algorithm that we put in place to get the rules that were holding at the time an event was logged t_{event} is shown in *Algorithm 1*. The inputs of the algorithm are the administrative log, the time when the investigated event happened (was logged), and the time of the investigation (now), and the output is the rules that were holding at the time of the event.

The first step consists of creating instances from the administrative log events that follow *Happens(e,t)* (line 2). Next, the *StartPermittedAdminActionQuery* and *EndPermittedAdminActionQuery* are SQWRL queries that allow extracting the *Initiates* and *Terminates* statements of the fluents *is-permitted(a,add_rule,r)* and *is-permitted(a,remove_ rule,r)*, and their times (lines 4-5).

As mentioned earlier, the SQWRL queries can only work on known individuals. Since the inferred axioms and SQWRL results are not written back to the ontology, and the successful addition of a rule depends on administrative rights (*c.f (5.4) and (5.5)*), we force the assertions of the *HoldsFor* axioms by calling the *AssertHoldsForStatement* method (line 6). This is useful to have correct results in the next query.

Algorithm 1 Find Holding Rules

Input: *AdministrativeLog*, t_{event}, t_{invest}

Output: *Rules that hold at* t_{event}

1: $HoldsAt \leftarrow \emptyset$
2: *CreateIndividuals(AdministrativeLog)*
3: *runSWRLRules()*
4: *runSQWRLQuery(StartPermittedAdminActionQuery)*
runSQWRLQuery(EndPermittedAdminActionQuery)
5: $StartPermittedAdminActionResult \leftarrow getSQWRLResult(StartPermittedAdminActionQuery)$
$EndPermittedAdminActionResult \leftarrow getSQWRLResult(EndPermittedAdminActionQuery)$
6: *AssertHoldsForStatement(StartPermittedAdminActionResult, EndPermittedAdminActionResult,* t_{invest}*)*
7: *runSQWRLQuery(StartRuleQuery)*
runSQWRLQuery(EndRuleQuery)
8: *StartRuleResult* $\leftarrow$ *getSQWRLResult(StartRuleQuery)*
EndRuleResult $\leftarrow$ *getSQWRLResult(EndRuleQuery)*
9: *AssertHoldsForStatement(StartRuleResult,EndRuleResult,*t_{invest}*)*
10: *runSQWRLQuery(HoldsForQuery)*
11: *HoldsForResult* $\leftarrow$ *getSQWRLResult(HoldsForQuery)*
12: **while** *HoldsForResult.next()* **do**
13: *Fluent* $\leftarrow$ *HoldsForResult.getValue("f")*
14: *Rule* $\leftarrow$ *HoldsForResult.getValue("r")*
15: *Policy* $\leftarrow$ *HoldsForResult.getValue("p")*
16: $t_1 \leftarrow$ *HoldsForResult.getValue("*t_1*")*
17: $t_2 \leftarrow$ *HoldsForResult.getValue("*t_2*")*
18: **if** $t_1 < t_{event} <= t_2$ **then**
19: *HoldsAt.add(new HoldsAt(Fluent,Rule,Policy,*t_{event}*))*
20: **end if**
21: **end while**
22: **return** $HoldsAt$

The *AssertHoldsForStatement* function is shown in *Algorithm 2*.

Algorithm 2 AssertHoldsForStatement

Input: *SQWRLResult result1, SQWRLResult result2,* t_{invest}

```
1:  while result1.next() do
2:      StartTimeValues ← ∅
        EndTimeValues ← ∅
3:      Fluent1 ← result1.getValue("f")
4:      t1 ← result1.getValue("t")
5:      ontology.addAxiom(hasFluent, HoldsFor, Fluent1)
6:      StartTimeValues.add(t1)
7:      while result2.next() do
8:          Fluent2 ← result2.getValue("f")
9:          t2 ← result2.getValue("t")
10:         if Fluent1 == Fluent2 && t2 > t1 then
11:             EndTimeValues.add(t2)
12:         end if
13:     end while
14:     if EndTimeValues.size() > 0 then
15:         t' ← min(EndTimeValues)
16:         ontology.addAxiom(hasStartTime, HoldsFor, t1)
            ontology.addAxiom(hasEndTime, HoldsFor, t')
17:     else
18:         t' ← max(StartTimeValues)
19:         ontology.addAxiom(hasStartTime, HoldsFor, t')
            ontology.addAxiom(hasEndTime, HoldsFor, tinvest)
20:     end if
21: end while
```

Algorithm 2 creates the corresponding individuals of the respective classes, e.g., *HoldsFor, Fluent, etc.*, and assigns the correct values to the data property assertions *hasStartTime* (t_1) and *hasEndTime* (t_2). For each fluent, it applies the minimum value of t_2 that is greater than t_1, and if no value of t_2 was found, it assigns t_{invest} *(now)*.
Continuing in *Algorithm 1*, the same steps are done for the fluents $r_i \in p_1$ (lines 7-9).

Once all the *HoldsFor* axioms are asserted, another query is executed (*HoldsForQuery*) to get the rules that held at t_{event} (lines 10-22). Finally, this process can be done for each log event in the regular log as shown in *Algorithm 3*.

For each access time, the holding rules are fetched. After that, the values of the subject, object, and environmental attributes that were in place at the time of the logged event are collected using the *GetHoldingAttributesValue(event)* function. This is where the *Mediator Agent* starts searching for other agents that have the required attributes as explained in *Chapter 4*. Next, line 10 executes a query to see if the event was permitted or not at the time of its execution according to expression (5.3). Lines 11-27 allow us to run through all the valid rules at the time of the access. If at least one rule is matched, then no violation is returned. If all the rules result in a violation, that means that in none of the cases, the event has appeared to be permitted; hence; a violation is returned according to (4.7). It is worth mentioning that administrative violations are also obtained by executing *ViolationQuery* as they are also deduced through axiom (4.7).

Algorithm 3 Violation Detection

Input: *RegularLog*, *AdministratieLog*, t_{invest}

Output: *Violations*

for each *event* in *RegularLog* **do**
  violation ← ∅
  t_{event} ← *event.getTime()*
  HoldingRules ← *FindHoldingRules(AdministrativeLog,* t_{event}, t_{invest}*)*
  NumberOfViolatedRules = 0
  for each *rule* in *HoldingRules* **do**
    for each *attribute* in *rule* **do**
      GetHoldingAttributesValues(event)
    end for
    HoldsAtPermittedUserEvent ← *getSQWRLResult*(*HoldsAtPermittedUserEventQuery*)
    if *HoldsAtPermittedUserEvent.isEmpty()* **then**
      NumberOfViolatedRules++
      Continue
    else
      break
    end if
  end for
  if *NumberOfViolatedRules == HoldingRules.size()* **then**
    AssertNotHoldsAtStatement(event,t_{event}*)*
  end if
  violation ← *getSQWRLResult(ViolationQuery)*
  return *violation*
end for

5.7 Experimentation

To evaluate the performance of our approach, we consider the extraction of different numbers of events that occurred during a specific period. Moreover, taking into account the administrative policy, we vary the number of rules that held during the

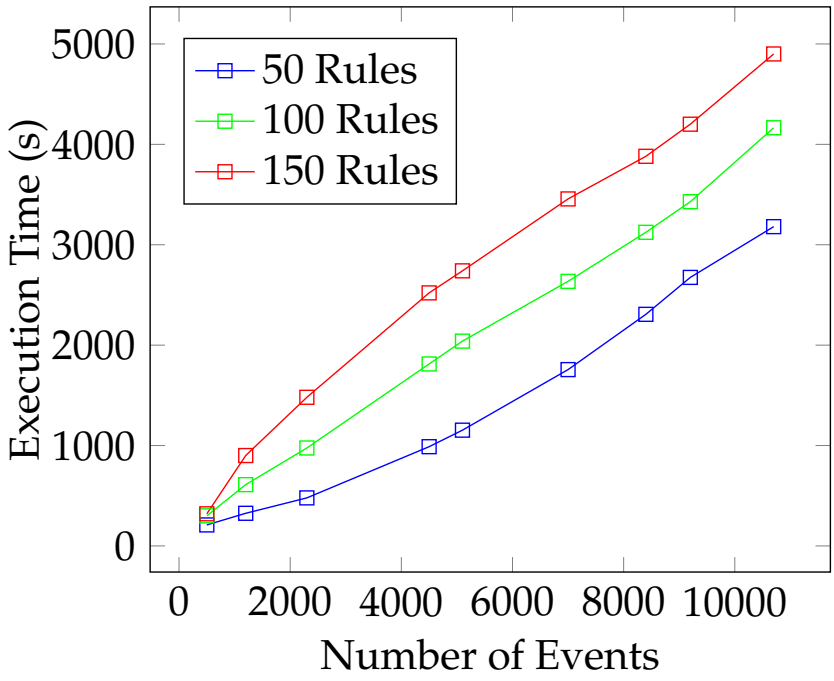
Table 5.1: The considered violation rate of each tested number of events

Number of Events	Considered Violation Rate
500	2%
1200	5%
2300	10%
4500	15%
5100	15%
7000	20%
8400	20%
9200	20%
10700	25%

interval of time of each batch of events. We also suppose that a certain percentage of these events does not respect the security policy; hence, it constitutes a violation. *Table 5.7* shows the considered violation rate for each number of events. It is also worth mentioning that we use the same tools as the ones presented in *Section 4.6.1*.

In contrast, we keep the same rule order when changing the number of holding rules. We recall that when an event matches at least one rule of the security policy, it is considered as a no violation. Thus, the order in which the rules are being verified influences the time at which the verification process of an event will stop. For instance, even if 50 rules were holding at a specific time, the verification will stop at the 35^{th} defined rule if the concerned event matches it. That being said, looking that when increasing the number of events, the number of violations is also increased (*c.f. Table 5.7*), the execution time will increase as well since all the rules are checked in case of a violation. Similarly, when the number of holding rules increases for the same number of events, the execution time follows the same trend. Given that the order of rules is kept unchanged, the resulting time variation is caused by the violating events as they impose verifying all the rules. The obtained results are shown in *Figure 5.5*.

Figure 5.5: Time in function of number of events



5.8 Conclusion

In this chapter, we treated the a posteriori access control in case of an administrative policy. To the best of our knowledge, it is the first contribution of this kind since none of the previous works took into account the evolution of security rules over time when checking policy compliance. The proposed violation detection mechanism was based on the deductive Event Calculus and SWRL. Besides, we modelled the ABAC administrative policy according to AMABAC, and we enhanced its expressiveness by allowing the specification of conditions when assigning permissions to administrators. Moreover, we gave expressions that show the relation between a logged event and the rules that held at the time of the access to detect violations, as well as the inter-dependency between administrative actions and the valid security rules. Thus, the proposed approach also permits the assurance of the conformity of administrative actions as these latter can constitute violations too.

After detecting violations, responsibilities should be fixed and decisions should

be made about whether sanctions should be applied or not. This process of setting responsibilities falls under the third component of the a posteriori access control that is very important since it is the one that deters users from committing violations. However, setting responsibilities is a difficult problem, especially when considering the policy administration rules. Therefore, we treat accountability in the a posteriori access control in the next chapter.

Chapter 6

Accountability in the A Posteriori Access Control

6.1 Introduction

It has been shown that the preventive access control can be inadequate in environments where exceptions may occur, making the a posteriori access control more suitable.

Moreover, we recall that the a posteriori access control is composed of three critical components that are logging, auditing, and accountability. In the previous chapters, we treated the first two components and we showed how logging serves as evidence in case of a suspicious violation as it traces users' actions that are done in the system. Furthermore, we proposed a framework to analyze logs in the auditing process to check their consistency and compliance with the defined security policy. As for accountability, several definitions were given to it since it is used broadly in a variety of fields. For instance, [83] defined accountability as the "right of some actors to hold other actors to a set of standards, to judge whether they have fulfilled their responsibilities in light of these standards, and to impose sanctions if they determine that these responsibilities have not been met." [121] gave a more technical definition for accountability that is "Accountability is the ability to hold an entity, such as a person or organization, responsible for its actions." [68] called an entity "accountable with respect to a policy" that is if, whenever the entity violates the policy, then with some positive probability it is, or could be, punished.

Thus, regardless if sanctions will be actually applied or not, we can all agree that "*accountability is a way to deter the user from committing violations*" as it constitutes a threat of punishment that pressures the user psychologically.

A good number of researches treated the a posteriori access control by focusing mainly on its first two components that are concerned in detecting violations, but unfortunately they underestimated the importance of developing an accountability mechanism. Nevertheless, the a posteriori access control must be combined with a dissuasive sanction and reparation policy so that users are not tempted to violate the security policy. In this chapter, we define a framework for accountability to decide whether the user should be sanctioned or not, *once a violation is detected*. In particular, we show how accountability can be deployed as a requirement and as a mechanism in the a posteriori access control. We also treat both cases of a static security policy and an administrative security policy in which the blame can be passed to the administrator.

6.2 Accountability: a Requirement and a Mechanism

As mentioned earlier, in the a posteriori access control, users' actions are monitored to assure their compliance with the security policy. We recall that the security policy of an information system corresponds to a set of rules defining access control requirements (permissions, prohibitions) as well as usage control requirements (obligations) relating to the actions that a user carries out in this information system. This policy can be modeled according to different access control models such as RBAC[70], ABAC[94], OrBAC[62], etc. Thus, when a user performs an action that is not conform with the rules defined in the security policy, the action is considered a violation. However, the flexibility that offers the a posteriori access control allows having certain exceptions for which the actions of users become permitted, or the user becomes blameless. We thus, define a violation as follows:

Definition 6.2.1. ***(Violation)***
A violation is an event, that is an action *op* done by a subject *u* on an object *o* at a specific time, that abuses the security policy *without taking in consideration the exceptions*.

It is also worth to mention that we consider, in this chapter, that the accountability

process starts ***after a violation is detected***.

In this connection, security analysts can derive different conclusions when analyzing access logs:

1. The concerned subject did not violate the security policy, in this case the problem would arise either from errors in system functions or from external malice.
2. The subject has violated the security policy but there are legitimate reasons which justify this behaviour and which invalidates this violation but does not exclude the responsibility of the subject without sanctioning him.
3. The subject has violated the security policy but no mitigating circumstances could be determined, he is then responsible and punishable for his unauthorized action.

Even if in an a posteriori environment the user is trusted, there is always a motivating reason that convinces him to breach the law to serve his self-interest and access data. Therefore, it is evident that leaving the access open to users exposes the system to different security threats that can be internal/external, malicious/non malicious, intentional/accidental [111], and that can cause severe consequences such as fraud, disclosure of sensitive information, destruction of information, etc. Since the a posteriori access control is based on a trustworthy environment in which users are knowledgeable of their rights (in reality, users are usually notified of their responsibilities and validate them by signing a confidentiality charter), we consider that the detected access policy violations are internal and intentional. Thus, the first possibility of the violation being caused externally is eliminated. Now that access policy violations are presumed, decisions should be made to determine whether the violator should be punished or not.

In contrast, the accountability framework can be seen in two different angles:

1. It can be considered as a set of requirements (a theory) that should be employed in the system to enforce the deterrence of policy violations.
2. It can be thought of as a mechanism to define and apply sanctions when violations are committed.

Moreover, it has been argued about whether increasing the probability of punishment is more effectively deterrent than an increase in the severity of punishment [75]. It all depends on whether the person who is tempted to violate the policy is a risk lover or not. In the following we discuss the requirements that should be adopted to deter policy violations and we propose an accountability mechanism *in case of the a posteriori access control*.

6.2.1 Accountability as requirement

Deploying measures that increase the users' perception of accountability in the information system will likely make the users experience systematic processing and awareness which will increase conformance with the policy. In [200], the authors presented an accountability theory to reduce access policy violations through system artifacts and showed how this theory could increase accountability perception. We thus recall the three discussed system dimensions that heighten accountability perception that are *identifiability, evaluation, and social presence*. *Identifiability* ensures that user's actions can be linked to him/her while *evaluation* assesses his actions according to some normative ground rules and with some implied consequences. As for *social presence*, it assumes that user's performances can be seen by others.

Indeed, these three criteria are assured in the a posteriori access control. First, logging makes sure that all accesses can be traced; hence, their subjects can be identified. In addition, the monitoring and auditing that is done by analyzing logs evaluate the conformity of these accesses with the security policy. Finally, although it is not always the case where one can see an other's actions especially in environments in which sensitive data is involved, the administrator or the auditor can always have a peek regardless if he is performing monitoring or not.

Having these three requirements in the information system will decrease the user's intent to commit access policy violations. However, unexpected circumstances could happen which will force the user to perform an unauthorized action or have an exceptional access. To take into consideration these latter, we consider a fourth requirement that is the *justification obligation*.

Definition 6.2.2. ***(Justification Obligation)***

A *justification obligation* is an *obligation* that states that in case of exception (e.g., emergency) that pushes the user to perform an action that is outside his sphere of access, the user must declare his access with a *justification*.

Definition 6.2.3. ***(Justification)***
A *justification* is the reason (purpose) for which the user has performed an unauthorized access. It is denoted as *j=(u,op,o,r,t)*, where *r* is the reason for which the user *u* performed an unauthorized action *a* on the object *o*, and which is logged at time *t*.

Moreover, each access event can only have one justification.

In contrast, most of the times, in case of a sudden emergency, the user does not have the time to justify his action before doing it. Therefore, we consider that the *justification obligation* should be done *a posteriori* during a certain period of time after the access. This time period is usually defined by the organization. Besides, we consider that once the justification is logged, it cannot be modified later on.

At this point some might be wondering how this requirement will enforce the deterrence of policy violations. In fact, not respecting this obligation is a violation by itself; hence, the probability of applying punishments will increase and risk-taking will decrease. Now that we enforced the deterrence of the a posteriori policy violation, we should integrate this requirement in the a posteriori access control.

6.2.2 Accountability as a mechanism

After what has been discussed in the previous sections, a user is held accountable in the a posteriori access control once he/she violates the security policy. To be more specific, in the accountability process, the user is questioned to justify his actions. Interestingly, it is common to distinguish the implication of *responsibility* when defining *accountability*. However, the concept of *responsibility* can have different meanings [42]. Therefore, we define responsibility in the a posteriori access control as follows:

Definition 6.2.4. ***(Responsibility in the a Posteriori Access Control)***
A user is *responsible* for his/her actions and their consequences, if he/she violates the security policy.

This definition of responsibility is logical with regard to the possibilities of conclusion presented in *Section 6.2*. Thus, being responsible is independent of the punishments.

On the other hand, *accountability* not only regards the *responsibility*, but also the *liability* of accesses performed by a user in an information system.

Definition 6.2.5. ***(Liability)***
A user is *liable* if he/she is responsible, and should be blamed and sanctioned for his/her undesirable actions.

Since we assumed that the accountability process starts once violations are detected, the user is always responsible. Nevertheless, we distinguish, in the following, the cases in which the user is liable; hence, should be punished.

The case of a Static Policy

In this section, we treat the case of a static security policy that is not subject to modifications as in *Chapter 4*.

To start with, we impose a *justification obligation* to be logged by the user when he performs an unauthorized access. Thus, the non-existence of this justification is a violation of the obligation, and appoints no reason to invalidate the committed violation. Therefore, a user is considered *liable* if he violated the security policy and did not justify his violation as follows:

$$violation(u, op, o) \wedge \neg \exists j[justification((u, op, o), j)] \rightarrow is\text{-}liable(u, op, o) \tag{6.1}$$

While it is certain that the user is liable in the above circumstance (violation of the *justification obligation*), it is not the case when a justification exists. As a matter of fact, other factors should be taken into account:

- The reason of access provided in the justification should be categorized as an “allowed exception".
- The justification should be “honest".

In contrast, in traditional RBAC or ABAC access control models that are not leveraged to dynamically adapt to fringe cases [5, 65], exceptions are not encoded. Therefore, we consider a particular setting, where an exception policy, that specifies how the rights of users to access resources are affected in various exceptional situations, complements the security policy. The exception policy is generally a less constraining version of the security policy. For example, in a hospital, an access control policy specifies that each doctor has access to the medical records of his/her own patients. However, if a patient has a heart attack, then any doctor in the ward can have access to that patient's medical record during this emergency. *Figure 6.1* shows our a posteriori access control setting. We also consider that the exception policy is static and not subject to changes.

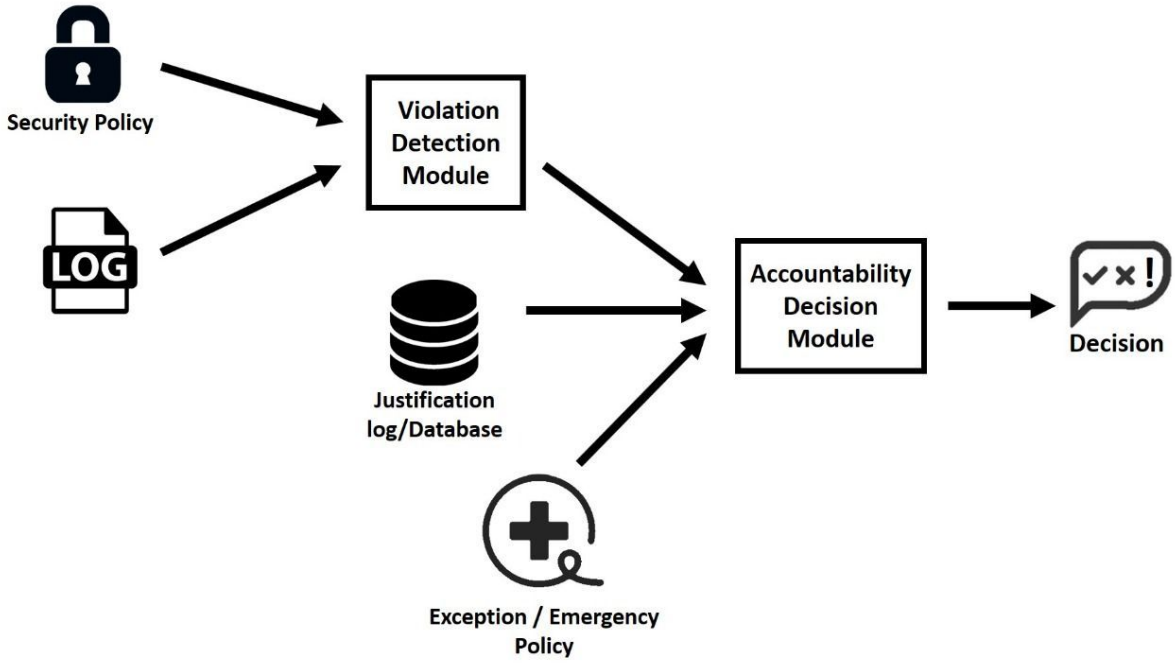


Figure 6.1: A Posteriori Access Control with Accountability

In this respect, a justification is considered to be valid if the reason provided in it, is relevant to the permissions defined in the exception policy. We refer to [32, 5] for inferring the relevance between an access permission and a purpose. Therefore, a user

is also liable if he/she provided an invalid justification:

$$violation(u, op, o) \wedge is\text{-}invalid(justification((u, op, o), j)) \rightarrow is\text{-}liable(u, op, o) \tag{6.2}$$

Moving on to deciding if the justification is honest or not, the problem becomes more difficult. It must be pointed out that the honesty of a justification is investigated only if this latter is valid. As previously mentioned, the user will have a limited time period after his/her exceptional access to justify it. Moreover, it has been shown in [184] that users tend to lie when they are pressured in time and are more likely to be honest when they have enough time to answer when they are being interrogated. In consequence, the time period chosen by the organization should have a reasonable length but should not either be so long so that the user will not have the time to plan a lie. Nevertheless, the user might sometimes justify his exceptional access after the defined time period because he/she had successive emergencies or simply because he/she forgot to do so. Therefore, we distinguish between an *onTimeJustification*, and a *lateJustification* that we define as follows:

Definition 6.2.6. ***(onTimeJustification)***
An *onTimeJustification* is a justification that is logged during the required time period.

Definition 6.2.7. ***(lateJustification)***
A *lateJustification* is a justification that is logged after the required time period.

That being said, we consider that when a user logs an *onTimeJustification*, he/she is being honest. This assumption was made since the user provided a valid justification in the right time; hence, he/she is respecting the security rules. On the other hand, qualifying a *lateJustification* can be confusing as it can be the object of a malicious (dishonest) user and a non-malicious (honest) one. To solve this problem, we examine the *impact* or the *damage* (e.g., data destruction) that results following an exceptional access in the information system. Thus, a *lateJustification* is considered to be dishonest if there is an impact on the system. Besides, when a *lateJustification* is provided with no impact, the concerned user will be given a *warning* while always being responsible for

his action:

$$violation(u, op, o) \wedge \exists j[is\text{-}valid(lateJustification((u, op, o), j))] \wedge \\ \neg\exists i[impact((u, op, o), i)] \rightarrow is\text{-}responsible(u, op, o) \wedge warning(u, w) \tag{6.3}$$

Furthermore, if a user receives more than n warnings, then he is classified as malicious; in other words, his $(n+1)^{th}$ late justification is dishonest (even if it is honest, he did not respect the *justification obligation* several times). n is also defined by the organization. This condition was put to not oppress the user in case he is being honest even though he violated in a way the *justification obligation*. The *warning* will give him the chance to adapt his behavior in the future; hence, will serve as a reminder to respect the obligation. Nevertheless, n should not take a great value so that the probability of being sanctioned remains high (ideally should be equal to 1 or 2). The steps over which the accountability decision model reasons is depicted in *Figure 6.2*.

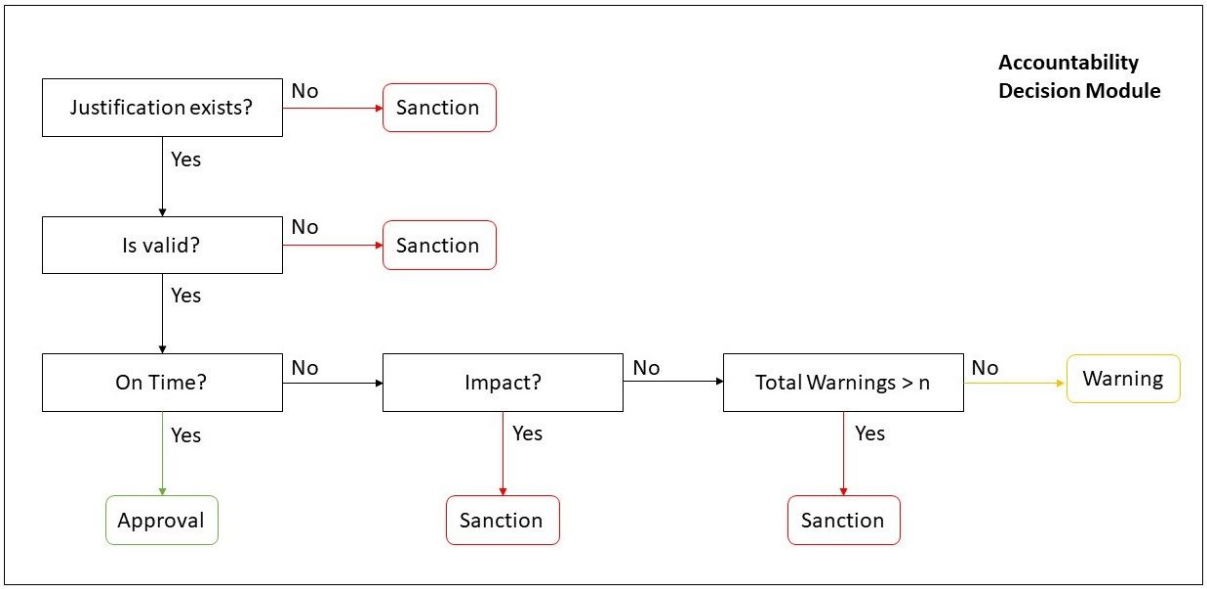


Figure 6.2: Accountability Decision Module in case of a Static Policy

As a consequence, we define the profile of a *sanctionable* user as a user who did not provide a justification, or provided an invalid justification, or provided a valid *lateJustification* and his unauthorized access had an impact on the system, or got $n+1$

warnings:

$$\begin{aligned} &\textit{is-sanctionable}(u) \equiv \\ &\quad \neg\exists[\textit{justification}((u, op, o), j)] \vee \\ &\quad \textit{is-invalid}(\textit{justification}((u, op, o), j)) \vee \\ &\quad \exists j[\textit{is-valid}(\textit{lateJustification}((u, op, o), j))] \wedge \exists i[\textit{impact}((u, op, o), i)] \vee \\ &\quad \textit{totalWarnings}(u, n+1) \end{aligned} \tag{6.4}$$

As a result, the user will be liable, if he provoked a violation and he is sanctionable:

$$\textit{violation}(u, op, o) \wedge \textit{is-sanctionable}(u) \rightarrow \textit{is-liable}(u, op, o) \tag{6.5}$$

We can notice that (6.1) and (6.2) can be derived from (6.5).

Once the decision has been made about the user's accountability, sanctions and remedies should be applied. When thinking of sanctions, we first imagine an amount of money. Therefore, we consider a *sanction S* as a *penalty* that is calculated based on whether the user is sanctionable or not. The value of the penalty is chosen by the auditing authority (for example, it can be equal to the salary of the employee). However, it must be noted that different types of sanctions can be considered such as getting fired, prison, etc. We define γ as a boolean variable that indicates if the user is sanctionable or not. Thus, $\gamma = 1$ ($\bar{\gamma} = 0$) if the user is sanctionable and 0 otherwise. In consequence, the sanction value can be calculated as follows:

$$S = \textit{penalty} \times (1 + \gamma - \bar{\gamma}) \tag{6.6}$$

In addition, other remedies can be put in place such as taking away the right of "breaking the glass", that is the ability to perform prohibited actions when necessary. This remedy will be adopted when the organization looses the confidence she had in the user, after this latter had caused several violations and been given multiple sanctions.

The case of an Administrative Policy

As presented in Chapter 5, administrators can also be held accountable following their actions. Normally, they are responsible of creating the security rules that permit or prohibit regular users from performing an access. Moreover, in order to do a "break-glass" action, the user might ask the administrator to create him/her a specific rule to perform the action. The administrator can also create/remove rules on his/her own without prior demand from the user. Whatever the reason for the rule's creation/removal is, the rules should be appropriate, and the administrator should not abuse his/her rights. We thus, consider the same setting represented in *Figure 6.1*, but this time the security policy can be changed over time by administrators. Nevertheless, the exception policy remains static. That being said, a security auditor s can blame the administrator, with respect to a *justification*, without exempting the user of his/her responsibilities. It is worth noting that the security auditor must be different than the concerned administrator so that the accountability decision will not be biased. That being said, the user remains responsible since even if it was the administrator's fault, he is the one who performed the unauthorized action; hence, participated in the violation. In this case, the user will be given a warning , and the administrator is held responsible too:

$$\begin{aligned} &violation(u, op, o) \wedge \exists j[justification((u, op, o), j)] \wedge blame(s, (a, op, r)) \\ &\rightarrow is\text{-}responsible(u, op, o) \wedge warning(u, w) \wedge is\text{-}responsible(a, op, r) \end{aligned} \tag{6.7}$$

In contrast, a new regulation came into force in May 2018, that is the General Data Protection Regulation (GDPR) [202]. GDPR requires the collected data to be used only for *specific purposes*. Therefore, [18] proposed a framework to design access control policies in reference to the legal environment of the GDPR. In consequence, we suppose that when an administrator has the right to create/remove/modify a specific rule, his/her action leads to a GDPR compliant Access Control Policy (ACP), enforcing the principle of data protection by design and by default. In consequence, when an administrator is blamed for his/her actions or when he simply commits a violation, the first thing to check if the resulted ACP from performing the action is GDPR compliant. If it is not the case, the administrator is liable and should be sanctioned.

On the other hand, if the resulting ACP is GDPR compliant, the process returns to the normal liability check. In this connection, the *justification obligation* is also imposed on the administrators. Thus, the same conditions are applied to have a sanctionable administrator (*c.f. (6.5)*). As a result, an administrator is liable for performing an operation on a security rule, if the resulting ACP is not GDPR Compliant or if he is sanctionable as follows:

$$\begin{aligned} &violation(a, op, r) \wedge [\neg GDPRCompliant(a, op, r) \vee is\text{-}sanctionable(a)] \\ &\rightarrow is\text{-}liable(a, op, r) \end{aligned} \tag{6.8}$$

The functioning of this new version of the accountability decision module is shown is *Figure 6.3*.

Moving on to calculating the sanction's value, it is the same as in equation (6.6). Nevertheless, when the ACP is not GDPR compliant, the sanctions will be set according to the GDPR, that is 4% of the total global annual turnover or 20 million euros, whichever is the higher. The GDPR's fine is normally imposed by authorities on the company. However, the organization can also charge the administrator, as he is the one representing it, and the value of the sanction is normally made precise in a previously established agreement.

6.3 Conclusion and Future Work

In this chapter, we proposed a framework for accountability in the a posteriori access control. We showed how accountability can be seen as a requirement and a mechanism, and how integrating the *justification obligation* in the process can increase the probability and the severity of sanctions. Besides, we formalized our approach using the descriptive logic. However, all the given expressions can be easily expressed in SWRL.

Moreover, we addressed the accountability problem in two cases. A static policy is used in the first case, in which only regular users are held responsible. In the second case, an administrative policy is deployed, opening the possibility of blaming the administrator for his/her actions. Thus, both the user and the administrator are

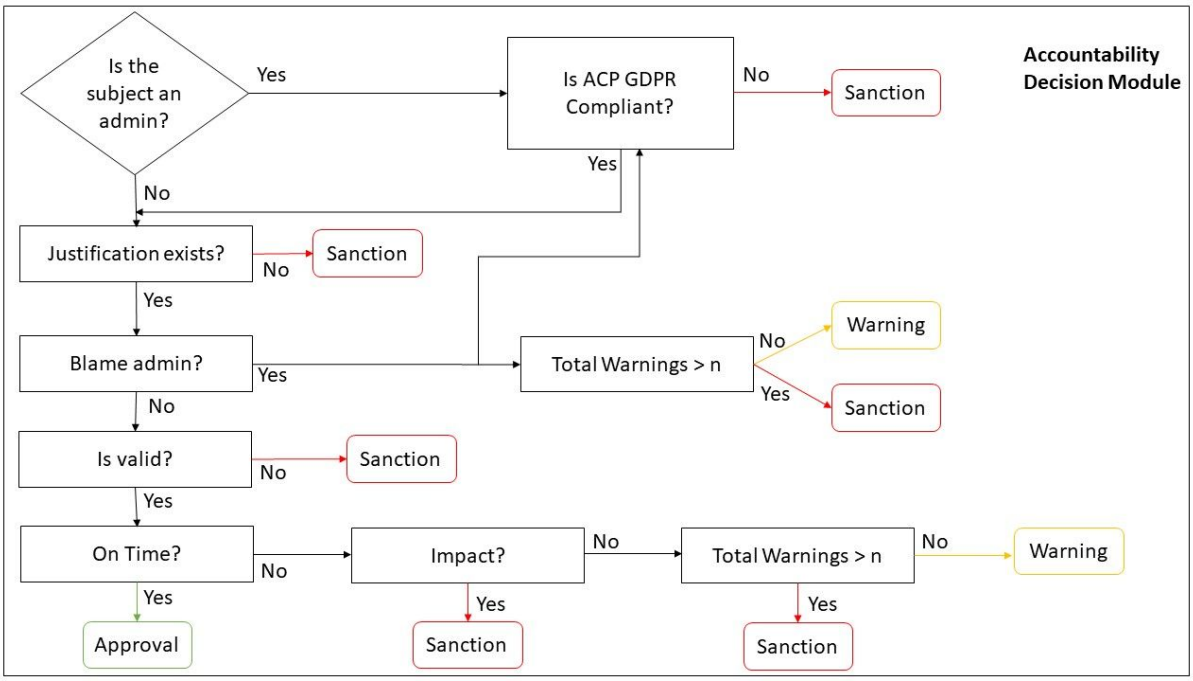


Figure 6.3: Accountability Decision Module in case of an Evolutive Policy

held responsible. Furthermore, we integrated GDPR compliance in the second case, as administrators usually represent organizations; hence, sanctions can be applied by both the organization and GDPR authorities.

Chapter 7

Conclusions and Perspectives

The main objective of this work was to propose a framework to perform the a posteriori access control that detects potential violations of the security policy. The concerned mechanism is a monitoring process that is based on logs as they provide evidence of users' actions. Moreover, users are deterred from committing policy violations by enforcing the principle of applying sanctions and remedies. It has been also discussed that this type of access control is divided into three components that are *log processing*, *log analysis*, and *accountability*. We have covered these three areas of the a posteriori access control, introduced some new aspects that were not treated previously in the literature, and provided novel solutions.

To start with, the first component involves extracting useful information from logs. Nevertheless, these latter can be found in multiple places, and can be generated from different log sources, which leads to having a variety of log formats and contents. Thus, to dissolve this heterogeney of log formats, we proposed to use a semantic mediator that is based on query rewriting to extract information. We proved that this approach has a lot of advantages especially that it is economical in terms of processing, as log formats remain intact and transformations are only done on the queries. Besides, we showed how it allows us to extract information in terms of the security policy by defining one global ontology with the concepts *subject*, *action*, *object*, and *timestamp*. These concepts also constitute the standard information that can be found in any log.

Focusing on log analysis, we addressed this step by taking into consideration some vital factors. First of all, we supposed that in case of an expressive security policy

(e.g., ABAC), the extracted log information is not sufficient to evaluate its conformance with the security policy. Thus, this information needs to be semantically enrich with other attributes for a correct analysis. As data can be distributed in the information system, we automated this information collection task by proposing a multi-agent system architecture, and detailed the function of each agent.

In contrast, we leveraged the a posteriori access control to include policy temporal compliance that takes into account the possible changes that attributes might undergo over time. Thus, we formalized the violation detection mechanism using the Event Calculus, and implemented it in SWRL. The Event Calculus was chosen as an appropriate basis to formalize our problem since both logs and policy modifications are event-driven. Besides, we integrated this temporal verification in the proposed multi-agent system to have a uniform framework. As a result, the investigation consisted not only in checking if the users and objects have the right attributes to perform an action, but if they had them *at the right time*.

On the other hand, we treated log analysis and the policy temporal compliance in both cases of static and evolutive security policy. We recall that by static we refer to the expression of the security policy and not its application. In the first case, the temporal verification only concerned the attributes of the logged events, while in the second case it went beyond attributes to include security rules. That being said, we considered having an administrative security policy in which an administrator can change the rules defined in the security policy, as well as the violations that can be caused by administrators. Always using the Event Calculus, we showed how the policy verification in this case can be recursive; hence, we defined a stopping condition to define the initial state of the process and guarantee its termination.

Moving on to the accountability component, we have argued how accountability can be seen as a requirement and as a mechanism in the a posteriori access control. We presented the *justification obligation* requirement and showed how it can be used in a mechanism to decide the liability of users based on an exception policy. We also treated the accountability problem in case of static and administrative policy. Moreover, we included a module that verifies the GDPR compliance of the resulting policy following an administrative action to increase the severity of punishment.

Although our research lead to several contributions in the field of access control, particularly, in the a posteriori access control, several open issues and perspectives can be treated in the future since as we all say: "there is always room for improvement". Therefore, we present these perspectives in the following.

7.1 Perspectives

Given the multidisciplinary nature of the theories and techniques used in this thesis, we did not have the opportunity to explore some aspects of our proposal in greater depth. In the following, we discuss some limitations of our proposal and discuss how the contributions presented in this thesis can be extended.

7.1.1 Log Analysis with Incomplete Information

In our approach, we have considered that all the needed attributes and information can be found in the information system as taking decisions about the legitimacy of the executed actions requires having all the information. In consequence, one limitation of our approach is the unavailability of the needed information. One missing attribute, that can be due to a source breakdown, not functioning agent, or simply not logged information, etc., can disrupt the violation detection mechanism.

[176] proposed an approach for access control under uncertainty, where users can afford the cost of the permission. However, the cost is calculated based on probabilities, which cannot be applicable in case of an a posteriori access control, where decision is binary and applying sanctions is involved. Consequently, the use of the abductive reasoning could prove itself as a good solution to solve this problem, as it permits to determine the assumptions that are missing to reach the conclusion. Moreover, we plan to provide an accountability solution when the violation is indecisive.

7.1.2 Policy Conflict Resolution

It was shown that our model considers that there is no violation if the logged event matches at least one rule in the policy. Thus, our approach does not take into

consideration policy conflicts [206], particularly, the case where the same logged event and attributes lead to two different decisions. Thus, we would like to take into account in the future, the resolution of the conflicts and redundancies that may exist between the rules for a better violation detection. This problem can be delicate in the case of the a posteriori access control since removing redundancy from access control policies requires minimizing the number of authorizations in the policy itself, which would not provide the needed flexibility and would lead to a higher number of violations [86]. Moreover, it is important to consider the context when resolving policy conflicts [143, 187] to decide which rule is more powerful and should be applicable when performing the a posteriori analysis.

7.1.3 Combining a Priori and a Posteriori Access Control

Another future objective is to be able to have the a priori and the a posteriori controls cohabit in the same system, and to be able to switch certain controls from an a priori mode to an a posteriori mode (or vice versa) according to the evolution of the trust granted to the user. To do this, a model of the evolution of the trust granted to a user should be defined according to the actions he performs in the system and the possible violations of the security policy perpetrated by this user. Thus, when a user has the required trust level, he will have the option to "break the glass" that activates the a posteriori access control. It is also important to integrate this trust model in the expression of the security policy to specify the deployment of certain security rules based on the trust granted to the user.

7.1.4 Contextualizing the Exception Policy

In our accountability framework, we considered both cases in which the expression of the security policy can be static or subject to changes using an administrative model. Nevertheless, we did not take into consideration the evolution of the exception policy that can also change depending on the context. For instance, in case of a crisis, access permissions are updated assuring the validity of a higher number of justifications [189]. In fact, finding valid justifications when treating the violations a posteriori would allow us to enrich and contextualize the exception policy. Therefore, we would

like to treat the changes of this policy along this contextualization process that will influence the applicability of sanctions.

7.1.5 Considering Usage Control Requirements and Obligations

To have a framework that unifies the required elements of both traditional access control models and trust management, Usage Control (UCON) has been proposed [159]. It considers particular missing components of traditional access control, such as the concepts of obligations and conditions. While these obligations can be set a priori, they should be performed a posteriori by users after accessing a resource. Nevertheless, there is no guarantee that these obligations will be fulfilled even if they are attached to an emergency policy. Therefore, it is important to extend the current work to include a posteriori usage control enforcement [154].

7.2 Raising Awareness Among Organizations

7.2.1 Context

It is evident that access control is vital for organizations to protect their most valuable asset, that is data. Therefore, a company must choose the most convenient access control model and mechanism for its business. We have provided, in the previous chapters, solutions for the different components of the a posteriori access control. All these propositions were inspired from facts that are present in a real organization.

In this section, we provide some use cases that were offered by a real organization and show how the nature of data can influence and block the a posteriori access control. Therefore, we expose uncovered problems regarding logs and security policies that confirm the validity of the hypothesis that we took in this work and to raise awareness among organizations.

7.2.2 The Organization's Business

The organization that provided us the use cases is a trusted third party that ensures the validity, compliance, and balance between different parties and the exchanges and commitments. Its main clients are healthcare professionals and companies. Moreover, it acts as a digital agent on behalf of its clients by guaranteeing them control of their data in a regulatory and risk-free technical context in which they cannot operate alone. Therefore, the organizations possess important data centers with a large volume of sensitive data.

7.2.3 Use Cases

In this section, we present three log analysis use cases that had an intention to perform an a posteriori access control. Nevertheless, different problems arose among these cases that made the policy compliance evaluation incomplete. We thus, present the results to raise awareness among companies. We recall that in order to perform an a posteriori access control, both logs and the security policy should be present, and must be relevant to each other. It must also be noted that we do not show the provided data for confidentiality issues.

Use Case 1

In the first instance, we received two sysmon log files [192] from the Security Operations Center (SOC) of the organization. The first one shows the authenticated users, while the second represents the processes that were executed by users. Moreover, each file contained almost 1700 events. We thus, focused on the file showing the processes as it represents users' actions in the organization. The users are the company's employees and the logs reflect the processes that they have executed to complete their jobs. The information included in the sysmon processes files is: *timestamp*, *event_id*, *action*, *host_name*, *user_name*, *user_domain*, *process_name*, *process_guid*, *process_id*, *process_integrity_level*, *process_parent_name*, *process_parent_guid*, and *process_parent_id*. It is also worth to mention, that these sysmon logs were transmitted to an ELK platform [63]. Therefore, we received them structured in a table.

These logs were easily modelled in an ontology, and mapped to our proposed global ontology as the *user_name* is the *subject*, the *action* is the *action*, *process_name* is the *object*, and the *timestamp* is the *timestamp*. Nevertheless, we could not have the security policy of the organization as it was not stated in a proper document. We only could have some complementary information, but not the policy itself. For instance, we received a file that contains the list of authorized softwares. Each software was added to this list following an employee's request that is logged via a ticketing tool. After verification, all the softwares that were used in the logs were authorized in the list. However, the users who executed these processes were not the same as the ones who created the tickets. On the other hand, when a request is critical, the employee is invited to fill an exemption form to have a temporary administrative account. Unfortunately, we could neither access the roles of the employees, nor the reason why they need a specific tool.

Clearly, the information provided in these files was not easy to analyze as the access/security policy was missing. Some inputs, such as the list of authorized users, are needed to understand the log and discover deviations. This lead to many unanswered questions, from which we mention: Once a software is authorized for a user and added to the list, will it be authorized to all other users? Who are the people concerned in validating tickets and exemptions? Which rules/criteria do they follow to take a decision of validating (or not) a ticket? Etc.

In consequence, since policy conformance checking requires an existing a priori process model (security policy) that we do not have, our analysis was restrained on discovering the process model itself. Thus, we applied process mining techniques [198] on these logs to see if we can deduce a general model of what is going on in the enterprise.

Process mining is a field of data mining, which consists of building and analyzing business processes based on event logs. One big advantage of process mining techniques is that the information is compiled objectively. This means that process mining techniques capture what is actually happening in an organization, and not what one might think it is happening. In process mining, each event log refers to a case, an activity, and a point in time. An event log can be seen as a collection of cases,

and a case can be seen as a trace/sequence of events. Besides, the tests were done using ProM [199], the most well-known and popular process mining tool.

In contrast, we applied several process mining algorithms such as Heuristics Miner [207], and Fuzzy Miner [87]. Both Heuristics and Fuzzy Miners' results were "spaghetti-like", showing all the details without distinguishing what is important and what is not. This is due to the nature of the log file that we have, as every action done by every employee on his PC is being registered in the logs. For example, if an employee clicks five times on firefox.exe, each click will be considered as an event log and the action would be "processcreate". There was no defined cycle or process model that can be deduced from these log files, since the users were not doing a specific "process". They were just executing the applications needed for their work to be done. That is why, the discovered process models were not abstract at all.

In consequence, to perform an a posteriori access control, we were interested in having log files that are generated from a specific application, rather than the logs produced by the employees' actions in the whole organization. Moreover, it is more likely to find a well defined security policy in an application domain.

Use Case 2

Since many flaws appeared in the first case (no security policy, no process in logs), we attempted to analyze logs that are generated from a specific application. The first application that we considered was related to the medical domain. It is used by internal employees as well as external clients to handle medical charges and refunds.

In this case, we succeeded in getting the security policy even if it was not up to date. It was modeled according to RBAC, where each user is assigned to a group in which several roles are defined and associated with different permissions. These implemented roles and permissions were deduced from the needs of each job position. Neither static nor dynamic separation of duty was specified, and the principle of least privilege was not applied. Besides, this policy contained some imperfections. For instance, some accesses were defined but did not have any role assignment. Moreover, constraints were implemented by restricting access to a group of users without being explicitly defined in the policy, and prohibitions were managed by making the profiles

inactive (access by suspending rights).

Moving on to the logs, we have noticed a significant issue. The logs of the concerned application are neither structured nor have a specific format. It seems that whichever flow that passes through the application is logged in the same log file. For instance, the file contains both XML and text logs simultaneously. However, the actions executed in the application were not entirely logged. We could not identify the user who is performing the action, as well as when he logged in/logged out from the application; hence, we could not distinguish when the user changes, making it impossible to reconstitute the session of the user. This was very problematic since even if the logs contained useful (sensitive) data, it was hard to understand what was going on. There were also logins that appeared in the logs that were not defined in the policy. This might be because the provided policy lacked of updates.

For all the reasons discussed above, we could not perform the a posteriori access control in this application domain, especially because logs did not have a specific format (regardless if they were structured or not), and the user's session was impossible to reconstruct. Therefore, we headed over another application.

Use Case 3

The second application that we dealt with was about handling alerts that are triggered from a fraud detection application. This latter was an IBM proprietary so we could not access the rules based on which an alert is raised.

The alerts application generated Apache HTTP logs that reflected the actions executed by the security team of the organization. These logs were also saved and visualized under ELK; hence, they were clean and structured. Moreover, they contain the usual http fields that are: *time*, *authenticatedUserId*, *method*, *url*, *status*, and *userAgent*. The *authenticatedUserId* presents the *subject*, the *method* presents the action, the *url* presents the *object*, and the *time* presents the *timestamp*.

On the other hand, the security policy of this application is modelled according to RBAC. It defines 3 roles, and the permissions assigned to these roles were composed of an action and a url path. We also received another file that contained complementary

information of the policy such as the users to roles assignments.

In contrast, the url paths contain some variable fields which values varied from one logged event to another. These variable fields, represent certain attributes (domain, alert type, alert status, etc.), which values are restricted depending on the group of users. Nevertheless, these restrictions are not explicitly defined in the security policy but directly implemented in the application. We believe that adding these attributes to the security policy can contextualize it and enrich it to transform the policy from RBAC to ABAC. Moreover, the relation between each action and url path is bijective. Thus, even if the logs show http methods (GET, POST, PUSH, etc.), we could deduce the real action that was done. Nevertheless, the users (*authenticatedUserId*) are represented with a Universally Unique Identifier (uuid), imposing the need to have a decryption key to get the real user/username. Unfortunately, we could not have this key. Therefore, we could not identify the users who are performing the actions to perform an a posteriori access control. In consequence, we decided to do Policy Engineering, also called Role Engineering in case of RBAC, to see if we can improve the existing policy.

Role Engineering helps in implementing an RBAC model by ensuring that all users possess relevant permissions to execute their designated tasks. It must be error-free to prevent unauthorized accesses. Therefore, the role creation process attempts to ensure that only required permissions are made available to the concerned users. In addition to creating a set of roles, role engineering can also take into account several constraints and determine a hierarchy among the roles.

Two approaches can be adopted when performing role engineering: (1) Top-Down, and (2) Bottom-Up. The difference between the two approaches resides in the basis from which the role creation process starts. For instance, in the Top-Down approach, the process begins by analyzing the structure of the organization to identify the business processes that constitute its workflow, while the Bottom-Up approach starts at the permission level by considering the existing permission assignments of the users of the organization. It is evident that the Bottom-Up approach is best suited in our case since we are starting from the existing permissions. Therefore, we used a very well-know Bottom-Up technique that is Role Mining [128, 197], and which goal is to deduce the user-role assignment (UA) and role-permission assignment (PA) relations from the

existing user-permission assignment (UPA) relation. These relations are presented as boolean matrices. Nevertheless, our mining will be based on the generated logs and not on the existing policy. Thus, if every logged event is considered as a permission that was executed by a user, the size of the UPA matrix will be huge considering the log file size. In this respect, we reproduced the approach presented in [100].

The idea of [100] is that co-occurring permissions are likely to belong to the same role. Therefore, the authors assumed that roles are sets of permissions that appear together frequently in the logs. Moreover, their algorithm is considered as a special case of frequent itemset mining (FIM) [2]. However, a major difference between this approach and FIM is that there is no known value for minimum support since there is no a priori knowledge of how frequent a pattern should be to be considered frequent. Thus, [100] proposed a score-based mechanism in which a sorted list of all top-scored itemsets is kept, enabling the elimination of the lowest scored itemsets if memory concerns arise. Summarizing the proposed algorithm, it begins with breaking up the log entries of each user denoted as AHL_U. It then goes through each AHL_U and gathers neighboring permissions in form of candidate roles and assigns each role a degree of cohesion. Candidate roles are generated by enumerating possible sets of permissions, and the degree of cohesion is calculated based on the frequency of coincidence of the permissions of a candidate role. The degree of cohesion is used to determine how good a role is, and which roles are better. For reasons of practicality and memory usage, the algorithm maintains a large but fixed-size list of candidate roles and remove low scored roles when the list grows beyond this fixed size. Finally, members of each role are found and then a sufficient number of top-scored roles are selected to cover all permissions and users.

To test the above algorithm, we extracted 30 minutes of activity that is around 1000 events (we could not extract more at the time because of accessibility issues). In addition, there were 9 active users in the extracted log files. The algorithm took around 4 minutes to execute, and returned 30 roles. We recall that a role in role mining is a set of users that are associated to a set of permissions. We concluded that the obtained roles are sub-roles of the 3 initially defined roles, as these latter are very generic. Nevertheless, we could not validate conformance of the mined users' permissions with the ones defined in the security policy since, as mentioned earlier, we could not decrypt

the uuids representing the users in the logs.

7.2.4 Discussion and Perspectives

The three use cases that were provided, were problematic in different ways that prevented us from accomplishing the a posteriori access control. We thus, discuss some vital aspects that organizations should have to perform this kind of security check.

It was shown that controlling the correct application of the security rules defined in the policy is not always ensured in an organization, which can expose it to several frauds and attacks. When it is found, the security policy of an organization is defined in terms of the access requests that must be authorized. In order to set the security rules up and running, administrators implement them so that they can be interpreted by the machine. However, during the implementation phase, the organization may undergo several changes, and human errors may occur. For instance, some of the staff who is involved in implementing the policy may not be responsible or knowledgeable enough. Moreover, these errors can be classified into two types: incorrect authorizations, that represent access requests that are authorized by the implemented policy but should be prohibited, and incorrect denials represented by access requests that are not authorized by the implemented policy but should be allowed. This was demonstrated in *Use Case 2*, where some of the active users in the logs were not found in the security policy. Intuitively, incorrect authorizations are more difficult to detect and more problematic because they can be used to read confidential data and attack systems. On the other hand, incorrect denials are generally less complicated as when the user discovers that a valid access request is not authorized, he/she reports it to the administrator to modify what is implemented. Nevertheless, one should be aware that these modifications, which may take time to put in place, can result in a convoluted policy.

In addition, it has been proven that it is not overwhelming to not find a documentation of the security policy like in *Use Case 1*. This is because application designers and developers are usually more focused on the implementation. They do not necessarily document all the tasks performed relating to the security policy as they underestimate it. Therefore, it is crucial to have a well defined document that

describes the security policy. Moreover, a gap between the implemented security policy and the documented one (if it exists) can be carried out, making it essential to update the documentation every time a change in the implementation occurs, and to check their conformance from time to time. This was also highlighted in *Use Case 2*. Furthermore, as updating the security policy manually can be cumbersome, we recommended to use Role Engineering techniques to automate this task, by producing policies based on the implemented permissions and/or the organization's structure (Hybrid Role Engineering). In contrast, post-implementation control will be more relevant if expressive security policies such as ABAC or OrBAC are used. These models can be more complex than a simple RBAC model, where permissions are assigned to user roles, but better in expressing the context of the performed actions given the constraints that can be added regarding multiple attributes and environmental conditions. Thus, a lot of works were devoted to solve the problem of mining ABAC policies from logs [49, 212, 141]. However, the majority considered that the logs already contain all the attributes, which is not so realistic, or are augmented with complementary information which is more likely possible. For instance, we could complete the logs in *Use Case 3* with the restrictions values that were provided in a separate file. Yet, we noticed that it is more probable to find object attributes in the logs than subject attributes. Thus, we had this idea of mining ABAC policies after mining RBAC from logs [213]. The goal is to use only the logged data since complementary information may not be always found. That being said, a future perspective can be mining a Role-centric Attribute-based Access Control model (RABAC) from logs [108].

Another best practice is to ensure the Cloud Security Posture Management (CSPM) [44], that is "*a continuous process of cloud security improvement and adaptation to reduce the likelihood of a successful attack*". The role of CSPM solutions is to check misconfigurations in the cloud platform accounts and notify users of them as well as of the way to fix them. This makes CSPM very important since misconfigurations may lead to unwanted data breaches and leakage. Moreover, CSPM includes several security checks such as: "Identity, Security, and Compliance", "Monitoring and Analytics", "Inventory and Classification", and "Cost Management and Resource Organization". As more and more organizations move their data to the cloud, CSPM can be very helpful when performing the a posteriori access control since its inter-operate between

monitoring by offering detection, logging, and reports, and automation by addressing issues ranging from cloud service configurations to security settings that relate to governance, compliance, and security for cloud resources.

Moving on to the logs, the value of log analysis may differ depending on the context in which the investigation is being conducted. In the case of an a posteriori control, the aim will be to detect any potential violation of the security policy, in order to apply sanctions and/or reparation if necessary. Often forgotten, the analysis of logs is the first step of a good referencing, which content should be "deciphered". It is obviously a source of trust, offering honest data, considering that all accesses are logged. Unfortunately, the provided use cases showed that organizations might depreciate the importance of logs when designing a product, and generating logs is done only to log "something". Therefore, the critical aspect of lack of evidence (logs) must be addressed, especially in the case of fraudulent activities that perpetrated because of a fault in access rights and the inability to penalize the violator. To this end, it should be noted that in order to manage accesses properly, there must be a sufficient level of traceability for the analysis to be conducted correctly, as corrupted logs impact decision-making. It is also important to consider the quality of the information to be logged so that log files are understandable by auditors. Above all, it is not preferable to group all types of logs together in a single file, and the logged information should be more or less complete allowing at least the reconstruction of a user's session. This has been confirmed by the CNIL [45]: "*the logging must concern, at the very least, the accesses of the users, including their identifier, the date and time of their connection, the date and time of their disconnection*". In addition, it is for the best to log whether or not the requested action has been authorized by the system. Not logging everything is also required. Logs should be simple and useful, especially in terms of access control. As a good practice, the ANSSI has published security recommendations for the implementation of a logging system [182]. Besides, [31] proposed a log design for accountability.

Even if all the above requirements are assured, log analysis remains a difficult task because of the large volume of log files; hence, the need to process them. Furthermore, it stays challenging because logs are usually weakly structured, use a variety of formats and terminologies, and are spread over different files and systems. Thus, our proposition of using a semantic mediator to extract information from multiple

logs is justified. Moreover, the use of a multi-agent system to gather information and semantically enrich the extracted log information is appropriate since it is true that in reality, other useful information can be present somewhere else than logs (i.e., Use Cases 1 and 3).

To sum up, the main reason of the problems that we faced can be explained by the fact that we tried to perform an a posteriori analysis on a system that was enforced a priori. Consequently, the main components of the a posteriori access control, that are logs and the security policy, were not given the importance that they should have and presented clear weaknesses. Besides, the accessibility to data is being more and more difficult since the GDPR came on board. In fact, organizations are more aware of the importance of privacy when treating the data, resulting in anonymizing them when saved as in *Use Case 3*. In consequence, it is a matter to consider in the future the case of anonymous logs when performing the a posteriori access control, as they will certainly affect violations decisions.

Appendix A

French Summary: Analyse a Posteriori des Logs et Détection des Violations des Règles de Sécurité

A.1 Introduction

Adopter un mode de contrôle d'accès adéquat est essentiel pour que les organisations puissent garantir la confidentialité, l'intégrité et la disponibilité de leurs systèmes d'information. Les modèles de contrôle d'accès traditionnels vérifient les privilèges des utilisateurs avant de leur accorder l'accès aux ressources d'information. Cependant, il est indispensable de prendre en compte les usages et les pratiques de l'organisation, afin que la solution de sécurité déployée ne soit pas perçue comme une contrainte pour les utilisateurs avec un risque de rejet important. Par conséquent, dans certains environnements sensibles, tels que le domaine de la santé, où les utilisateurs sont généralement de confiance et où des évènements particuliers peuvent se produire, comme les situations d'urgence, les contrôles de sécurité mis en place dans les systèmes d'information correspondants ne doivent pas bloquer certaines décisions et actions des utilisateurs. Cela pourrait avoir des conséquences graves. À cet égard, il est important de pouvoir identifier et tracer ces décisions et ces actions afin de détecter d'éventuelles violations de la politique de sécurité mise en place et fixer les responsabilités. Nous considérons qu'une politique de sécurité d'un système d'information est un ensemble de règles définissant des exigences de contrôle d'accès (permissions, interdictions)

relatives aux actions effectuées par un utilisateur sur ce système d'information.

La problématique du contrôle d'accès a posteriori est assez récente, et consiste à surveiller les actions réalisées par les utilisateurs de manière à détecter les éventuelles violations de la politique de sécurité et appliquer des sanctions ou réparations. En général, ce processus de contrôle repose sur une analyse qui ne concernera que les logs relatifs à un sujet, une action ou un objet particulier en relation avec l'évènement déclencheur de l'investigation. En revanche, la principale référence de cette analyse a posteriori est la politique de sécurité, puisque l'objectif est de détecter les éventuelles violations de cette dernière. Néanmoins, le processus de contrôle des violations présente plusieurs difficultés qui doivent être surmontées.

Tout d'abord, la vérification des contenus des logs se fonde essentiellement sur l'expertise de la personne qui l'effectue et de ce qui lui semble utile d'investiguer. Elle peut aussi faire appel à une liste générique de contrôles de sécurité à effectuer qui n'est pas forcément adaptée au système cible. Il est donc nécessaire de définir un format de référence pour la politique de sécurité qui facilite la détection des violations potentielles. Par ailleurs, l'archivage et la journalisation ne sont pas toujours normalisés. Cela est généralement dû aux différents types de sources de logs, comme les serveurs d'application, les serveurs Web, les bases de données, etc. Ainsi, il est important de disposer d'un module qui permet d'extraire des informations pertinentes des fichiers journaux sans imposer un format particulier. En outre, le contrôle du respect de la politique a posteriori doit reposer sur des mécanismes de surveillance efficaces permettant de détecter les éventuelles violations. Il faut aussi qu'il soit associé à une politique de sanction et de réparation dissuasive pour que les utilisateurs ne soient pas tenter de violer la politique de sécurité.

Dans cette thèse, nous fournissons des moyens et de nouvelles solutions pour résoudre ces problèmes et améliorer le contrôle d'accès a posteriori. Premièrement, nous proposons une nouvelle méthode pour extraire des informations utiles des logs en termes de politique de sécurité. L'approche consiste à utiliser un médiateur sémantique qui réécrit les requêtes qui sont envoyées aux logs. Ensuite, nous traitons l'enrichissement sémantique des informations extraites des fichiers journaux, ce qui est utile dans le cas d'une politique de sécurité expressive, pouvant dépendre de

conditions contextuelles, où un bon nombre d'attributs, qui se trouvent dans des sources de données différentes, sont nécessaires pour définir les permissions d'accès. En plus, nous améliorons le contrôle d'accès a posteriori en ajoutant l'aspect temporel au processus de vérification de la politique. Comme les enquêtes sont menées a posteriori, nous soulignons qu'une évaluation correcte du respect de la politique ne doit pas se limiter au respect des attributs de sécurité requis, mais doit aller au delà pour vérifier que les attributs des entités concernées étaient valides au moment de l'accès. Dans un deuxième temps, nous traitons le cas où les règles définies dans la politique de sécurité peuvent changer au cours du temps, en utilisant un modèle administratif de la politique. De plus, nous considérons les violations qui peuvent être causées par les utilisateurs et les administrateurs, ce qui rend le processus d'évaluation récursif. Par suite, nous définissons une condition qui assure sa terminaison. Enfin, nous proposons un mécanisme d'imputabilité pour le contrôle d'accès a posteriori.

A.2 Extraction d'Informations des Logs à l'aide de la Médiation Sémantique

La première étape du contrôle d'accès a posteriori consiste à assurer la journalisation, c'est-à-dire le traitement des fichiers journaux. Cette étape est fondamentale car l'analyse sera basée sur les informations extraites. Toutefois, ce processus devient plus difficile avec la croissance du volume des données enregistrées. Pour cela, on propose une solution pour extraire des informations des logs en utilisant des techniques de médiation sémantique. L'objectif est de résoudre l'hétérogénéité entre les formats de log en provenance de différentes sources de log, vu que la médiation sémantique permet d'inter-opérer différentes sources d'information sans modifier leur fonctionnement interne. '

On définit un cadre particulier dans lequel nous déployons notre médiateur sémantique pour extraire des informations des logs en termes de politique de sécurité. On considère les éléments suivants :

- Il y a plusieurs sources de logs présentées par $S_1, S_2, ..., S_n$ et chaque source a un format particulier désigné par $f_1, f_2, ..., f_n$.

- La politique de sécurité est désignée par P et est représentée dans un modèle ontologique selon le modèle ABAC [94].
- Un médiateur sémantique existe entre la politique et les sources de logs pour le traitement des requêtes.
- Les logs fournis sont bien structurés pour permettre l'extraction des informations plus utiles [104].

Maintenant que nous avons défini notre paramètre d'extraction d'informations, nous considérons que les requêtes sont envoyées automatiquement de la politique de sécurité définie vers les logs. Le médiateur sémantique réécrira ensuite la requête exprimée sur un schéma source en une autre requête exprimée sur un schéma cible. Ce processus de réécriture est effectué en utilisant les correspondances sémantiques préalablement établies entre les différents schémas (ontologies dans notre cas).

Dans le médiateur, chaque source de logs est représentée par une ontologie. Il convient de mentionner qu'aucune source de logs ne sera modifiée. Ces ontologies locales servent en tant qu'un point d'accès conceptuel aux données lors des interactions entre le médiateur et les sources de logs. Ainsi, les logs ne seront pas transformés en ontologies, et les ontologies locales ne contiendront que les principaux concepts fournis par chaque source de logs, plutôt que les valeurs apparaissant dans chaque événement. De plus, une ontologie consensuelle est nécessaire pour représenter le domaine d'application. L'ontologie globale (également appelée ontologie de domaine), constitue le point d'entrée à partir duquel les requêtes envoyées aux sources de logs sont décomposées. Jusqu'à présent, il n'existe pas de format standard qui puisse représenter tous les types de logs, y compris les journaux d'application, car ces derniers sont généralement déterminés par le développeur du logiciel. Néanmoins, même si le contenu des logs peut varier considérablement d'une source à une autre, tous enregistrent simplement l'événement qui s'est produit, plus précisément, "qu'est-ce qui s'est passé ? quand ? et par qui ?". Par conséquent, les concepts qui forment l'ontologie globale sont les éléments essentiels de chaque log, qui sont le Sujet, l'Action, l'Objet et l'Horodatage. En plus, des correspondances sémantiques entre l'ontologie globale et les ontologies locales sont nécessaires pour réécrire la requête initiale en une union de requêtes correspondant à chaque source de log.

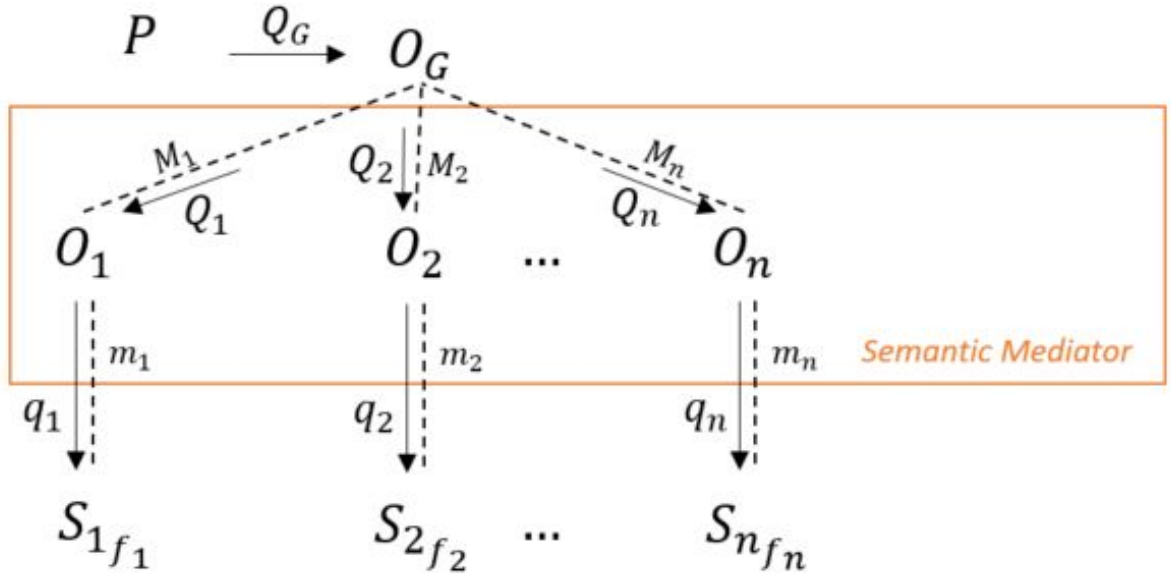


Figure A.1: Processus de réecriture de requêtes.

En outre, pour obtenir un résultat final unifié, on a divisé le processus de réécriture en deux étapes : la réécriture sémantique de la requête et la réécriture syntaxique de la requête.
La réécriture de la requête sémantique consiste à conserver le langage de la requête, tout en l'exprimant sur une autre ontologie source. Cela peut être fait en résolvant les correspondances qui existent entre deux ontologies. Pour l'instant, une requête SPARQL (langage de requêtes ontologiques) restera une requête SPARQL et la seule modification consistera à remplacer ses entités par leurs équivalents sémantiques. Dans la deuxième étape, une transformation syntaxique des requêtes SPARQL réécrites sera réalisée. La réécriture syntaxique consiste à modifier la syntaxe de la requête (le langage dans lequel elle est exprimée), tout en conservant sa sémantique. D'autre part, différents concepts peuvent être utilisés pour structurer les informations dans les fichiers journaux, tels que la relation dans le modèle relationnel, la balise XML, le CSV, etc. Ainsi, la requête SPARQL peut être convertie en une requête SQL, XQuery ou tout autre type de requête en fonction des formats de logs existants.
Le processus de réecriture de requêtes est montré dans la *Figure A.1*.

À partir des résultats obtenus après l'execution des requêtes, des axiomes et des assertions seront générés. Ainsi, pour contrôler l'accès, on précise si un sujet a

l'autorisation effectuer une action sur un objet. Etant donné qu'au niveau abstrait, l'expression de toute politique comprend un ensemble de quadruplets <sujet, action, objet, temps>, il est possible d'établir des liens entre les réponses des requêtes et les attributs de sécurité utilisés pour exprimer la politique de contrôle d'accès. On rappelle que l'expression d'une politique de sécurité présente les règles qui devraient être respectées, tandis que son application pourra dépendre notamment de la situation dans laquelle se trouvent les entités concernées.

En revanche, lorsque la politique de sécurité déployée est expressive, les informations extraites des logs nécessitent d'être enrichis sémantiquement. On traite ainsi ce problème dans la section suivante.

A.3 Détection a Posteriori des Violations

Après avoir extrait des informations des logs à l'aide du médiateur sémantique, le contrôle d'accès a posteriori passe à sa deuxième étape qui consiste à analyser ces informations pour détecter les violations de la politique de sécurité. En conséquence, une explication valable de la conformité des politiques doit exister, et la validité de cette explication repose sur la disponibilité des informations nécessaires à l'évaluation de la conformité des politiques.

Les modèles de politique de sécurité expressifs tels que RBAC [70], ABAC [94], OrBAC [62], etc., attribuent indirectement des autorisations aux utilisateurs par le biais de leurs attributs, et parfois aussi des attributs d'objet et des conditions contextuelles. En revanche, les logs ne permettent pas de retracer ce type d'informations en général. Il est donc nécessaire d'enrichir sémantiquement les données journalisées avec des informations complémentaires, de telle sorte que l'analyse des logs soit suffisamment précise pour prendre des décisions lorsque des violations sont commises.

Par contre, dans le contrôle d'accès a posteriori, de nombreux changements dans les attributs de sécurité peuvent avoir lieu entre le moment de l'accès et le moment de l'investigation, et les conditions contextuelles évoluent entre les accès également. Il est donc important de vérifier que les valeurs des attributs et les conditions d'accès étaient les mêmes que celles définies dans la politique de sécurité au moment où la ressource

d'information a été accédée. Ainsi, nous avons formalisé le mécanisme de détection des violations en utilisant l'Event Calculus (EC) [185] qui est un langage logique pour représenter et raisonner sur les événements et leurs effets, et on l'a mis en oeuvre en utilisant SWRL [93]. Le langage de l'EC contient: (1) un ensemble d'événements ou d'actions (2) un ensemble de fluents, c'est-à-dire un ensemble de propriétés dont les valeurs peuvent changer au cours du temps (et peut être vrai ou faux) (3) un ensemble de points de temps. Ces trois éléments sont essentiels et sont utilisés à travers des prédicats qui constituent le langage. Par conséquent, des axiomes peuvent être formées en reliant les différents prédicats pour décrire comment les événements et les fluents interagissent.

En outre, des changements peuvent également être apportés aux règles de sécurité, ainsi, la politique elle-même peut évoluer au fil du temps pour inclure des règles différentes. Ce type de modification peut être effectué à l'aide d'une politique de sécurité administrative. Il convient également de souligner que les administrateurs doivent avoir les bons privilèges pour pouvoir modifier la politique de sécurité. En conséquence, les actions administratives doivent également être contrôlées pour vérifier que les modifications appliquées sont aussi autorisées.

Pour résoudre ce problème, on propose une architecture de système multi-agents pour enrichir les logs sémantiquement, et on intègre la vérification temporelle dans ce système. On étudie aussi cettre vérification temporelle dans les deux cas où la politique de sécurité est statique et où elle peut évoluer conformément au modèle administratif.

A.3.1 Architecture Multi-Agents

L'objectif du système multi-agents proposé est de rassembler les attributs nécessaires à partir de différentes sources de données organisationnelles, et de vérifier si leur affectation temporelle est conforme ou non à la politique de sécurité. L'architecture de ce système est décrite dans la *Figure A.2*. Nous distinguons donc quatre types d'agents:

- **L'agent de la politique** s'occupe des règles définies dans la politique de sécurité. Il a la responsabilité de fournir les attributs nécessaires à l'agent médiateur.
- **L'agent médiateur** orchestre tous les messages échangés avec les autres agents

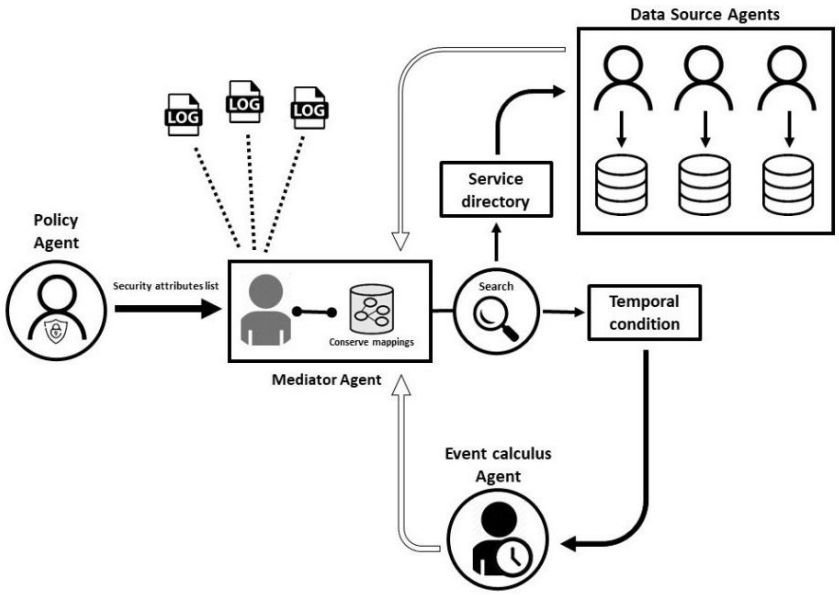


Figure A.2: Architecture du Système Multi-Agent

une fois qu'il a reçu la liste des attributs de l'agent de politique.

- **Les agents de sources de données** récupèrent les informations d'une source de données spécifique suite à une requête de l'agent médiateur.
- **L'agent de l'Event Calculus** vérifie les conditions temporelles définies dans la politique de sécurité en utilisant l'Event Calculus pour la représentation et le raisonnement sur les systèmes dynamiques.

A.3.2 Le Cas d'une Politique Statique

Lorsque les règles de sécurité sont statiques et ne changent pas avec le temps, cela signifie qu'à chaque fois qu'on consulte la politique de sécurité, les règles qui y sont définies restent les mêmes. Dans ce cas, l'analyse consiste à vérifer si la condition requise est maintenue au moment où un accès est effectué. Dans le modèle ABAC, la condition est constituée des attributs du sujet, des attributs de l'objet et des attributs de l'environnement (contexte). Ainsi, le processus d'évaluation consiste à trouver quels attributs et valeurs d'attributs étaient valides au moment de l'accès. Cette tâche est accomplie en collectant les valeurs des attributs en question, ainsi que les dates de leurs affectations à travers les différents agents, puis vérifier leur conformité temporelle avec

la politique de sécurité en utilisant l'Event Calculus et détecter les violations.

A.3.3 Le Cas d'une Politique Evolutive

Pour disposer d'un modèle de contrôle d'accès complet, il faut prévoir un modèle d'administration. Pour cela, le modèle ABAC a été récemment associé au modèle d'administration AMABAC [106]. Ce modèle contrôle qui a la permission d'attribuer/ révoquer des attributs et/ou des autorisations, et donc à modifier les règles de la politique de sécurité. Dans ce cas, on distingue deux types de vérification : (1) vérifier si l'action enregistrée au moment passé était autorisée ou non, en récupérant les règles qui étaient en place au moment où l'accès a eu lieu, et collecter les attributs définis dans ces règles, et (2) vérifier si les règles qui étaient déployées dans la politique au moment de l'accès ont été créées par les administrateurs qui avaient le droit de les créer. Vu qu'à chaque fois qu'on doit chercher les règles qui étaient valides au moment de l'accès, on doit vérifier si les administrateurs qui l'ont créées avaient l'autorisation de le faire, et à chaque vérification d'une règle, on doit vérifier que les administrateurs avaient le droit d'assigner les valeurs des attributs nécessaires, on entre dans un processus récursif. Pour arrêter cette récursivité, on définit une condition initiale qui impose d'avoir un seul administrateur qui a le droit de créer des règles et affecter les permissions aux autres administrateurs au moment de création de l'application. Une fois ces deux vérifications faites, les violations peuvent être déduites.

A.4 L'Imputabilité dans le Contrôle d'Accès a Posteriori

Une fois une violation détectée, le processus d'imputabilité commence. A cet égard, les analystes de sécurité peuvent avoir différentes conclusions: (1) le sujet concerné n'a pas violé la politique de sécurité, dans ce cas le problème proviendrait soit d'erreurs dans le fonctionnement du système, soit d'une malveillance externe, (2) le sujet a violé la politique de sécurité mais il existe des raisons légitimes qui justifient ce comportement et qui invalident cette violation, mais n'excluent pas la responsabilité du sujet sans le sanctionner, ou alors (3) le sujet a violé la politique de sécurité mais aucune circonstance atténuante n'a pu être déterminée, il est alors responsable et

sanctionnable pour son action non autorisée.

Etant donné que le contrôle d'accès a posteriori est basé sur un environnement de confiance dans lequel les utilisateurs connaissent leurs droits, nous considérons que les violations de la politique d'accès détectées sont internes et intentionnelles. Ainsi, la possibilité que la violation ait été causée de l'extérieur (e.g., usurpation d'idendité) est éliminée vu qu'on estime qu'il y aura suffisament de preuves qui disculpe l'utilisateur. Maintenant que les violations de la politique d'accès sont présumées, des décisions doivent être prises pour déterminer si le contrevenant doit être puni ou non.

En revanche, le cadre de responsabilité peut être vu sous deux angles différents :

1. Il peut être considéré comme un ensemble d'exigences (une théorie) qui devrait être utilisé dans le système pour appliquer la dissuasion des violations de la politique.

2. Il peut être considéré comme un mécanisme permettant de définir et d'appliquer des sanctions lorsque des violations sont commises.

A.4.1 L'Imputabilité en tant qu'Exigence

Le déploiement des mesures qui renforcent la perception de responsabilité des utilisateurs dans le système d'information permettra aux utilisateurs de faire l'expérience d'un traitement systématique et d'une prise de conscience qui augmentera la conformité avec la politique. Dans [200], les auteurs ont présenté une théorie de responsabilité pour réduire les violations de la politique d'accès par le biais d'artefacts du système et ont montré comment cette théorie pouvait augmenter la perception de responsabilité. Trois dimensions ont été identifiées pour assurer cette perception qui sont: l'identifiabilité, l'évaluation et la présence sociale. En effet, ces trois critères sont assurés dans le contrôle d'accès a posteriori, ce qui diminuera l'intention de l'utilisateur de commettre des violations de la politique d'accès. Cependant, des circonstances inattendues peuvent survenir qui forceront l'utilisateur à effectuer une action non justifiée ou à avoir un accès exceptionnel. Pour tenir compte de ces dernières, nous considérons une quatrième exigence qui est l'obligation de justification. L'obligation de justification est une obligation qui stipule qu'en cas

d'exception (par exemple, une urgence) qui pousse l'utilisateur à effectuer une action non autorisée, l'utilisateur doit déclarer son accès avec une justification qui contient la raison pour laquelle il a effectué cette action. Cette exigence permettra de renforcer la dissuasion des violations de la politique par le fait que le non-respect de cette obligation est une violation en soi.

A.4.2 L'Imputabilité en tant que Mécanisme

Les modèles traditionnels de contrôle d'accès, comme RBAC ou ABAC, ne sont pas exploités pour s'adapter dynamiquement aux cas marginaux car les exceptions ne sont pas codées. Par conséquent, nous considérons un cadre particulier, où une politique d'exception, qui spécifie comment les droits d'accès aux ressources des utilisateurs sont affectés dans diverses situations exceptionnelles, complète la politique de sécurité. La politique d'exception est généralement une version moins contraignante de la politique de sécurité. À cet égard, une justification est considérée comme valable si la raison qui la justifie est en rapport avec les autorisations définies dans la politique d'exception. La pertinence entre une autorisation d'accès et une justification peut être déduite d'après [32, 5]. La *Figure A.3* montre notre mécanisme d'imputabilité.

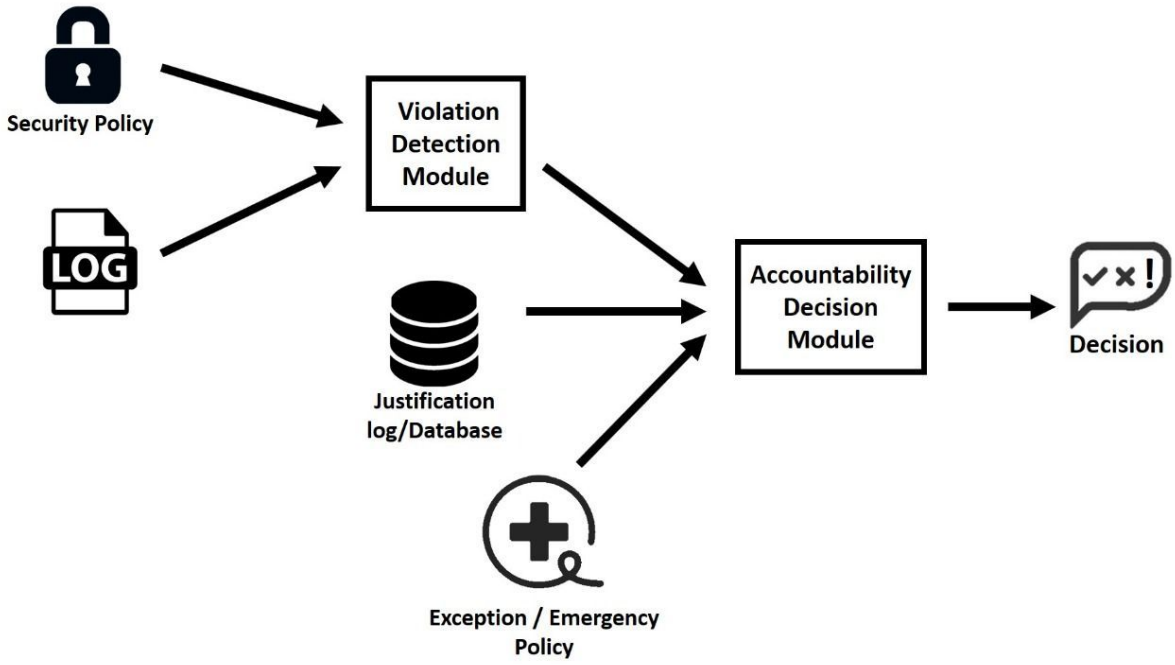


Figure A.3: Mécanisme d'imputabilité

On définit le profil d'un utilisateur sanctionnable comme un utilisateur qui n'a pas fourni de justification, ou a fourni une justification non valable, ou a fourni une justification valable et son accès non autorisé a eu un impact sur le système d'information.

Par ailleurs, en cas d'une politique évolutive, l'utilisateur pourra transférer sa responsabilité à l'administrateur. Ainsi, l'administrateur a besoin de justifier également ses actions.

A.5 Conclusion

L'objectif principal de ce travail était de proposer un cadre pour effectuer un contrôle d'accès a posteriori qui détecte les violations potentielles de la politique de sécurité. Le mécanisme concerné est un processus de surveillance qui repose sur l'analyse des logs pour fournir des preuves des actions des utilisateurs. En outre, les utilisateurs sont dissuadés de commettre des violations de la politique par le principe d'application

des sanctions. Ce type de contrôle d'accès est divisé en trois étapes: la journalisation, l'analyse des logs, et l'imputabilité. Nous avons donc couvert ces trois domaines du contrôle d'accès a posteriori, introduit de nouveaux aspects qui n'avaient pas été traités auparavant dans la littérature, et proposé de nouvelles solutions.

Tout d'abord, pour dissoudre l'hétérogénéité qui existe entre les différents formats des fichiers journaux, nous avons proposé d'utiliser un médiateur sémantique qui repose sur la réécriture de requêtes pour extraire des informations. Nous avons prouvé que cette approche présente de nombreux avantages, notamment qu'elle est économique en termes de traitement, étant donné que les formats de log restent intacts et les transformations ne sont effectuées que sur les requêtes.

Par ailleurs, en nous concentrant sur l'analyse des journaux, nous avons abordé cette étape en prenant en considération certains facteurs tel que le besoin de l'enrichissement sémantique des logs ainsi que la vérification temporelle. En effet, dans le cas d'une politique de sécurité expressive comme ABAC, les informations extraites des logs ne sont pas suffisantes pour évaluer leur conformité avec la politique de sécurité. Ainsi, ces informations doivent être enrichies sémantiquement avec d'autres attributs pour avoir une analyse correcte. Comme les données peuvent être distribuées dans le système d'information, nous avons automatisé cette tâche de collecte d'informations en proposant une architecture multi-agents et en détaillant la fonction de chaque agent. En revanche, nous avons exploité le contrôle d'accès a posteriori pour inclure la conformité temporelle de la politique qui prend en compte les changements possibles des attributs au fil du temps. Ainsi, nous avons formalisé le mécanisme de détection des violations à l'aide de l'Event Calculus qu'on a implémenté avec SWRL. Par conséquent, l'enquête ne consistait pas seulement à vérifier si les utilisateurs et les objets avaient les bons attributs pour effectuer une action, mais s'ils les avaient aussi au bon moment. D'autre part, nous avons traité l'analyse des logs et la conformité temporelle de la politique dans les cas où l'expression de la politique de sécurité est statique ou peut être modifiée au cours du temps en utilisant un modèle administratif. Ainsi, les violations peuvent être causées par les administrateurs également.

Concernant le processus d'imputabilité, nous avons expliqué comment la

responsabilité peut être considérée dans le contrôle d'accès a posteriori comme une exigence et comme un mécanisme. Nous avons présenté l'obligation de justification et montré comment elle peut être utilisée pour fixer la responsabilité des utilisateurs sur la base d'une politique d'exception. Nous avons également abordé ce problème en considérant l'expression statique et administrative de la politique.

Enfin, nous avons fourni trois cas d'usage qui ont été proposés par une organisation, et nous avons montré comment la nature des données peut influencer et bloquer le contrôle d'accès a posteriori. Par conséquent, nous avons exposé les problèmes découverts concernant les fichiers journaux et les politiques de sécurité afin de sensibiliser les organisations.

List of Publications

International Conferences

Farah Dernaika, Nora Cuppens-Boulahia, Frédéric Cuppens, and Olivier Raynaud. 2019. Semantic Mediation for A Posteriori Log Analysis. In Proceedings of the 14th International Conference on Availability, Reliability and Security (ARES '19), August 26-29, 2019, Canterbury, United Kingdom.

Farah Dernaika, Nora Cuppens-Boulahia, Frédéric Cuppens, and Olivier Raynaud. 2020. A Posteriori Access Control with an Administrative Policy. In Proceedings of the 2020 International Conference on Security and Management (SAM'20), July 27-30, 2020, Las Vegas, United States.

Farah Dernaika, Nora Cuppens-Boulahia, Frédéric Cuppens, and Olivier Raynaud. 2020. Accountability in the A Posteriori Access Control: a Requirement and a Mechanism. In Proceedings of the 13th International Conference on the Quality of Information and Communications Technology (QUATIC 2020), September 9-11, 2020, Online Conference.

Farah Dernaika, Nora Cuppens-Boulahia, Frédéric Cuppens, and Olivier Raynaud. 2020. A Posteriori Analysis of Policy Temporal Compliance. The 15th International Conference on Risks and Security of Internet and Systems (CRISIS 2020), November 4-6, 2020, Paris, France.

Bibliography

[1] Anas Abou El Kalam and Yves Deswarte. "Multi-OrBAC: A new access control model for distributed, heterogeneous and collaborative systems". In: *IEEE Symp. on Systems and Information Security (SSI 2006), Sao Paulo, Brazil*. 2006.

[2] Rakesh Agrawal, Ramakrishnan Srikant, et al. "Fast algorithms for mining association rules". In: *Proc. 20th int. conf. very large data bases, VLDB*. Vol. 1215. 1994, pp. 487–499.

[3] Kiyoharu Aizawa et al. "Efficient retrieval of life log based on context and content". In: *Proceedings of the the 1st ACM workshop on Continuous archival and retrieval of personal experiences*. 2004, pp. 22–31.

[4] Abdullah Alamri et al. "The mediator authorization-security model for heterogeneous semantic knowledge bases". In: *Future Generation Computer Systems* 55 (2016), pp. 227–237.

[5] Sandra Alves and Maribel Fernández. "A framework for the analysis of access control policies with emergency management". In: *Electronic Notes in Theoretical Computer Science* 312 (2015), pp. 89–105.

[6] *Apache Lucene*. `http://lucene.apache.org/`.

[7] Claudio A Ardagna et al. "Access control for smarter healthcare using policy spaces". In: *Computers & Security* 29.8 (2010), pp. 848–858.

[8] Mohamed Karim Aroua and Belhassen Zouari. "Modeling of A-Posteriori Access Control in Business Processes". In: *2012 IEEE 36th Annual Computer Software and Applications Conference Workshops*. IEEE. 2012, pp. 403–408.

[9] David Aumueller et al. "Schema and ontology matching with COMA++". In: *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. 2005, pp. 906–908.

[10] Hanieh Azkia et al. "A posteriori access and usage control policy in healthcare environment". In: *Journal of information assurance and security (JIAS)* 6.192 (2011), pp. 389–397.

[11] Hanieh Azkia et al. "Ontology based log content extraction engine for a posteriori security control." In: *Studies in health technology and informatics* 180 (2012), pp. 746–750.

[12] Hanieh Azkia et al. "Reconciling IHE-ATNA profile with a posteriori contextual access and usage control policy in healthcare environment". In: *2010 Sixth International Conference on Information Assurance and Security*. IEEE. 2010, pp. 197–203.

[13] Franz Baader et al. *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003.

[14] Leila Bahri, Barbara Carminati, and Elena Ferrari. "CARDS-collaborative audit and report data sharing for a-posteriori access control in DOSNs". In: *2015 IEEE Conference on Collaboration and Internet Computing (CIC)*. IEEE. 2015, pp. 36–45.

[15] Nathalie Baracaldo and James Joshi. "Beyond accountability: using obligations to reduce risk exposure and deter insider attacks". In: *Proceedings of the 18th ACM symposium on Access control models and technologies*. 2013, pp. 213–224.

[16] Daniel Barbara, Ningning Wu, and Sushil Jajodia. "Detecting novel network intrusions using bayes estimators". In: *Proceedings of the 2001 SIAM International Conference on Data Mining*. SIAM. 2001, pp. 1–17.

[17] Steve Barker. "Temporal Authorization in the Simplified Event Calculus". In: *Research Advances in Database and Information Systems Security*. Springer, 2000, pp. 271–284.

[18] Cesare Bartolini et al. "Towards a lawful authorized access: a preliminary GDPR-based authorized access". In: *14th International Conference on Software Technologies (ICSOFT 2019), Prague, Czech Republic*. 2019, pp. 26–28.

[19] D Elliott Bell and Leonard J LaPadula. *Secure computer systems: Mathematical foundations*. Tech. rep. MITRE CORP BEDFORD MA, 1973.

[20] Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing multi-agent systems with JADE*. Vol. 7. John Wiley & Sons, 2007.

[21] Walid Benghabrit et al. "Abstract accountability language". In: *IFIP International Conference on Trust Management*. Springer. 2014, pp. 229–236.

[22] Djamal Benslimane et al. "PAIRSE: a privacy-preserving service-oriented data integration system". In: *ACM SIGMOD Record* 42.3 (2013), pp. 42–47.

[23] C. Bertero et al. "Experience Report: Log Mining Using Natural Language Processing and Application to Anomaly Detection". In: *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*. 2017, pp. 351–360.

[24] Elisa Bertino, Piero Andrea Bonatti, and Elena Ferrari. "TRBAC: A temporal role-based access control model". In: *ACM Transactions on Information and System Security (TISSEC)* 4.3 (2001), pp. 191–233.

[25] Elisa Bertino et al. "GEO-RBAC: a spatially aware RBAC". In: *Proceedings of the tenth ACM symposium on Access control models and technologies*. 2005, pp. 29–37.

[26] Konstantin Beznosov. "Requirements for access control: US healthcare domain". In: *Proceedings of the third ACM workshop on Role-based access control*. 1998, p. 43.

[27] Khalid Zaman Bijon, Ram Krishman, and Ravi Sandhu. "Constraints specification in attribute based access control". In: *Science* 2.3 (2013), p. 131.

[28] Nikos Bikakis et al. "The SPARQL2XQuery interoperability framework". In: *World Wide Web* 18.2 (2015), pp. 403–490.

[29] Béatrice Bouchou and Cheikh Niang. "Semantic mediator querying". In: *Proceedings of the 18th International Database Engineering & Applications Symposium*. ACM. 2014, pp. 29–38.

[30] Achim D Brucker and Helmut Petritsch. "Extending access control models with break-glass". In: *Proceedings of the 14th ACM symposium on Access control models and technologies*. 2009, pp. 197–206.

[31] Denis Butin, Marcos Chicote, and Daniel Le Métayer. "Log design for accountability". In: *2013 IEEE Security and Privacy Workshops*. IEEE. 2013, pp. 1–7.

[32] Ji-Won Byun and Ninghui Li. "Purpose based access control for privacy protection in relational database systems". In: *The VLDB Journal* 17.4 (2008), pp. 603–619.

[33] Diego Calvanese et al. "Ontop: Answering SPARQL queries over relational databases". In: *Semantic Web* 8.3 (2017), pp. 471–487.

[34] James Cannady. "Next generation intrusion detection: Autonomous reinforcement learning of network attacks". In: *Proceedings of the 23rd national information systems security conference*. 2000, pp. 1–12.

[35] William B Cavnar, John M Trenkle, et al. "N-gram-based text categorization". In: *Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval*. Vol. 161175. Citeseer. 1994.

[36] Jan Cederquist et al. "The Audit Logic: Policy Compliance in Distributed Systems". In: *Zamm-zeitschrift Fur Angewandte Mathematik Und Mechanik* (Jan. 2006).

[37] Jan G Cederquist et al. "Audit-based compliance control". In: *International Journal of Information Security* 6.2-3 (2007), pp. 133–151.

[38] JG Cederquist et al. "An audit logic for accountability". In: *Sixth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'05)*. IEEE. 2005, pp. 34–43.

[39] Farid Cerbah. "Learning highly structured semantic repositories from relational databases". In: *European Semantic Web Conference*. Springer. 2008, pp. 777–781.

[40] Suresh Chari et al. "Ensuring Continuous Compliance through Reconciling Policy with Usage". In: *Proceedings of the 18th ACM Symposium on Access Control Models and Technologies*. SACMAT '13. Amsterdam, The Netherlands: Association for Computing Machinery, 2013, 49–60. ISBN: 9781450319508. DOI: 10.1145/2462410.2462417. URL: https://doi.org/10.1145/2462410.2462417.

[41] Pau-Chen Cheng et al. "Fuzzy multi-level security: An experiment on quantified risk-adaptive access control". In: *2007 IEEE Symposium on Security and Privacy (SP'07)*. IEEE. 2007, pp. 222–230.

[42] Laurence Cholvy, Frédéric Cuppens, and Claire Saurel. "Towards a logical formalization of responsibility". In: *Proceedings of the 6th international conference on Artificial intelligence and law*. 1997, pp. 233–242.

[43] D Clark and D Wilson. "A Comparison of Commercial andMilitary Security Policies". In: *IEEESymposium on Security and Privacy* (1987).

[44] *Cloud Security Posture Management - Gartner.* `https://www.gartner.com/en/documents/3899373/innovation-insight-for-cloud-security-posture-management`.

[45] *CNIL Logging Recommendations*. `https://www.cnil.fr/fr/securite-tracer-les-acces-et-gerer-les-incidents`.

[46] *Common Event Expression. White Paper.* `https://cee.mitre.org/docs/Common_Event_Expression_White_Paper_June_2008.pdf`.

[47] Rosaria Conte, Nigel Gilbert, and Jaime Simão Sichman. "MAS and social simulation: A suitable commitment". In: *International Workshop on Multi-Agent Systems and Agent-Based Simulation*. Springer. 1998, pp. 1–9.

[48] Ricardo Corin et al. "A logic for auditing accountability in decentralized systems". In: *IFIP World Computer Congress, TC 1*. Springer. 2004, pp. 187–201.

[49] Carlos Cotrini, Thilo Weghorn, and David Basin. "Mining ABAC rules from sparse logs". In: *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2018, pp. 31–46.

[50] Michael J Covington et al. "Securing context-aware applications using environment roles". In: *Proceedings of the sixth ACM symposium on Access control models and technologies*. 2001, pp. 10–20.

[51] Frédéric Cuppens and Nora Cuppens-Boulahia. "Modeling contextual security policies". In: *International Journal of Information Security* 7.4 (2008), pp. 285–305.

[52] Frédéric Cuppens and Alexandre Miège. "Administration model for or-bac". In: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer. 2003, pp. 754–768.

[53] Richard Cyganiak, David Hyland-Wood, and Markus Lanthaler. "RDF 1.1 Concepts and Abstract Syntax". In: *W3C Proposed Recommendation* (Jan. 2014).

[54] Mohamed Dahchour and Alain Pirotte. "The Semantics of Reifying n-ary Relationships as Classes." In: *ICEIS*. Vol. 2. 2002, pp. 580–586.

[55] Nicodemos Damianou et al. "The ponder policy specification language". In: *Policies for Distributed Systems and Networks*. Springer, 2001, pp. 18–38.

[56] Jérôme David et al. "The alignment API 4.0". In: *Semantic web* 2.1 (2011), pp. 3–10.

[57] Steven Dawson, Shelly Qian, and Pierangela Samarati. "Providing security and interoperation of heterogeneous systems". In: *Security of Data and Transaction Processing*. Springer, 2000, pp. 119–145.

[58] Hervé Debar, Marc Dacier, and Andreas Wespi. "Towards a taxonomy of intrusion-detection systems". In: *Computer networks* 31.8 (1999), pp. 805–822.

[59] Mari Antonius Cornelis Dekker and Sandro Etalle. "Audit-based access control for electronic health records". In: *Electronic Notes in Theoretical Computer Science* 168 (2007), pp. 221–236.

[60] Min Du et al. "Deeplog: Anomaly detection and diagnosis from system logs through deep learning". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 1285–1298.

[61] Andreas Ekelhart, Elmar Kiesling, and Kabul Kurniawan. "Taming the logs-Vocabularies for semantic security analysis". In: *Procedia Computer Science* 137 (2018), pp. 109–119.

[62] Anas Abou El Kalam et al. "Or-BAC: un modèle de contrôle d'accès basé sur les organisations". In: *Cahiers francophones de la recherche en sécurité de l'information* 1 (2003), pp. 30–43.

[63] *Elastic Search*. `https://www.elastic.co/fr/what-is/elk-stack`.

[64] Brendan Elliott et al. "A complete translation from SPARQL into efficient SQL". In: *Proceedings of the 2009 International Database Engineering & Applications Symposium*. ACM. 2009, pp. 31–42.

[65] Nada Essaouini et al. "Specifying and enforcing constraints in dynamic access control policies". In: *2014 Twelfth Annual International Conference on Privacy, Security and Trust*. IEEE. 2014, pp. 290–297.

[66] Sandro Etalle, Fabio Massacci, and Artsiom Yautsiukhin. "The meaning of logs". In: *International Conference on Trust, Privacy and Security in Digital Business*. Springer. 2007, pp. 145–154.

[67] Sandro Etalle and William H Winsborough. "A posteriori compliance control". In: *Proceedings of the 12th ACM symposium on Access control models and technologies*. 2007, pp. 11–20.

[68] Joan Feigenbaum. "Accountability as a driver of innovative privacy solutions". In: *Privacy and Innovation Symposium*. 2010.

[69] Christophe Feltus and Michaël Petit. "Building a responsibility model including accountability, capability and commitment". In: *2009 International Conference on Availability, Reliability and Security*. IEEE. 2009, pp. 412–419.

[70] David Ferraiolo, Janet Cugini, and D Richard Kuhn. "Role-based access control (RBAC): Features and motivations". In: *Proceedings of 11th annual computer security application conference*. 1995, pp. 241–48.

[71] Anna Ferreira et al. "How to break access control in a controlled manner". In: *19th IEEE Symposium on Computer-Based Medical Systems (CBMS'06)*. IEEE. 2006, pp. 847–854.

[72] Diogo R Ferreira and Lucinéia H Thom. "A semantic approach to the discovery of workflow activity patterns in event logs". In: *International Journal of Business Process Integration and Management* 6.1 (2012), pp. 4–17.

[73] Tim Finin et al. "R OWL BAC: representing role based access control in OWL". In: *Proceedings of the 13th ACM symposium on Access control models and technologies*. ACM. 2008, pp. 73–82.

[74] Jeffrey EF Friedl. *Mastering regular expressions*. " O'Reilly Media, Inc.", 2006.

[75] Lana Friesen. "Certainty of punishment versus severity of punishment: An experimental investigation". In: *Southern Economic Journal* 79.2 (2012), pp. 399–421.

[76] Seham Mohamed GadAllah. "The importance of logging and traffic monitoring for information security". In: *SANS reading room* (2003).

[77] Deepak Garg, Limin Jia, and Anupam Datta. "A Logical Method for Policy Enforcement over Evolving Audit Logs". In: *Computing Research Repository - CORR* (Feb. 2011).

[78] Meriam Ben Ghorbel-Talbi et al. "Delegation of obligations and responsibility". In: *IFIP International Information Security Conference*. Springer. 2011, pp. 197–209.

[79] Luigi Giuri and Pietro Iglio. "Role templates for content-based access control". In: *Proceedings of the second ACM workshop on Role-based access control*. 1997, pp. 153–159.

[80] Virgil D Gligor, Serban I Gavrila, and David Ferraiolo. "On the formal definition of separation-of-duty policies and their composition". In: *Proceedings. 1998 IEEE Symposium on Security and Privacy (Cat. No. 98CB36186)*. IEEE. 1998, pp. 172–183.

[81] Simon Godik and Tim Moses. "Oasis extensible access control markup language (xacml)". In: *OASIS Committee Secification cs-xacml-specification-1.0* (2002).

[82] Yoav Goldberg and Omer Levy. "word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method". In: *arXiv preprint arXiv:1402.3722* (2014).

[83] Ruth W Grant and Robert O Keohane. "Accountability and abuses of power in world politics". In: *American political science review* 99.1 (2005), pp. 29–43.

[84] Bill Gregg, Horacio D'Agostino, and Eduardo Gonzalez Toledo. "Creating an IHE ATNA-based audit repository". In: *Journal of digital imaging* 19.4 (2006), pp. 307–315.

[85] Thomas R Gruber et al. "A translation approach to portable ontology specifications". In: *Knowledge acquisition* 5.2 (1993), pp. 199–221.

[86] Marco Guarnieri et al. "On the notion of redundancy in access control policies". In: *Proceedings of the 18th ACM symposium on Access control models and technologies*. 2013, pp. 161–172.

[87] Christian W Günther and Wil MP Van Der Aalst. "Fuzzy mining–adaptive process simplification based on multi-perspective metrics". In: *International conference on business process management*. Springer. 2007, pp. 328–343.

[88] Phillip Hallam-Baker, Eve Maler, et al. "Assertions and protocol for the oasis security assertion markup language (saml)". In: *OASIS XML-Based Security Services Technical Committee* (2002).

[89] Hossein Hamooni et al. "LogMine: Fast Pattern Recognition for Log Analytics". In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. CIKM '16. Indianapolis, Indiana, USA: Association for Computing Machinery, 2016, 1573–1582. ISBN: 9781450340731. DOI: 10.1145/2983323.2983358. URL: https://doi.org/10.1145/2983323.2983358.

[90] Hossein Hamooni et al. "Logmine: Fast pattern recognition for log analytics". In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM. 2016, pp. 1573–1582.

[91] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[92] Matthew Horridge and Sean Bechhofer. "The owl api: A java api for owl ontologies". In: *Semantic web* 2.1 (2011), pp. 11–21.

[93] Ian Horrocks et al. "SWRL: A semantic web rule language combining OWL and RuleML". In: *W3C Member submission* 21.79 (2004).

[94] Vincent C Hu et al. "Guide to attribute based access control (abac) definition and considerations (draft)". In: *NIST special publication* 800.162 (2013).

[95] Vincent C Hu et al. "Guide to attribute based access control (abac) definition and considerations (draft)". In: *NIST special publication* 800.162 (2013).

[96] Neminath Hubballi et al. "An active intrusion detection system for LAN specific attacks". In: *Advances in Computer Science and Information Technology*. Springer, 2010, pp. 129–142.

[97] Koral Ilgun, Richard A Kemmerer, and Phillip A Porras. "State transition analysis: A rule-based intrusion detection approach". In: *IEEE transactions on software engineering* 21.3 (1995), pp. 181–199.

[98] *Implementing ArcSight CEF.* `https://www.secef.net/wp-content/uploads/sites/10/2017/04/CommonEventFormatv23.pdf`.

[99] Trent Jaeger. "On the increasing importance of constraints". In: *Proceedings of the fourth ACM workshop on Role-based access control*. 1999, pp. 33–42.

[100] Mohammad Jafari et al. "Role mining in access history logs". In: *Journal of Information Assurance and Security* 38 (2009).

[101] Radha Jagadeesan et al. "Towards a theory of accountability and audit". In: *European Symposium on Research in Computer Security*. Springer. 2009, pp. 152–167.

[102] *JAVA APIs for Date and Time*. `https://docs.oracle.com/javase/8/docs/api/java/time/package-summary.html`.

[103] Ahmad Javaid et al. "A deep learning approach for network intrusion detection system". In: *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*. 2016, pp. 21–26.

[104] Dileepa Jayathilake. "Towards structured log analysis". In: *2012 Ninth International Conference on Computer Science and Software Engineering (JCSSE)*. IEEE. 2012, pp. 259–264.

[105] Sadhana Jha et al. "An administrative model for collaborative management of ABAC systems and its security analysis". In: *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*. IEEE. 2016, pp. 64–73.

[106] Sadhana Jha et al. "Security analysis of ABAC under an administrative model". In: *IET information security* 13.2 (2018), pp. 96–103.

[107] Xin Jin, Ram Krishnan, and Ravi Sandhu. "Reachability analysis for role-based administration of attributes". In: *Proceedings of the 2013 ACM workshop on Digital identity management*. 2013, pp. 73–84.

[108] Xin Jin, Ravi Sandhu, and Ram Krishnan. "RABAC: role-centric attribute-based access control". In: *International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security*. Springer. 2012, pp. 84–96.

[109] Karen Sparck Jones. "A statistical interpretation of term specificity and its application in retrieval". In: *Journal of documentation* (1972).

[110] James BD Joshi et al. "A generalized temporal role-based access control model". In: *IEEE Transactions on Knowledge and Data Engineering* 17.1 (2005), pp. 4–23.

[111] Mouna Jouini, Latifa Ben Arfa Rabai, and Anis Ben Aissa. "Classification of Security Threats in Information Systems." In: *ANT/SEIT* 32 (2014), pp. 489–496.

[112] VVRPV Jyothsna, VV Rama Prasad, and K Munivara Prasad. "A review of anomaly based intrusion detection systems". In: *International Journal of Computer Applications* 28.7 (2011), pp. 26–35.

[113] Lalana Kagal. *Rei : A Policy Language for the Me-Centric Project*. Tech. rep. HP Labs, 2002. URL: `http://www.hpl.hp.com/techreports/2002/HPL-2002-270.html`.

[114] Alan H Karp, Harry Haury, and Michael H Davis. "From ABAC to ZBAC: the evolution of access control models". In: *Journal of Information Warfare* 9.2 (2010), pp. 38–46.

[115] Karen Kent and Murugiah Souppaya. "Guide to computer security log management". In: *NIST special publication* 92 (2006).

[116] Thanh Tran Thi Kim and Hannes Werthner. "An Ontology-based Framework for Enriching Event-log Data". In: *The Fifth International Conference on Advances in Semantic Processing*. 2011, pp. 110–115.

[117] Satoru Kobayashi, Kensuke Fukuda, and Hiroshi Esaki. "Towards an NLP-based log template generation algorithm for system log analysis". In: *Proceedings of The Ninth International Conference on Future Internet Technologies*. 2014, pp. 1–4.

[118] Teuvo Kohonen. "Self-organized formation of topologically correct feature maps". In: *Biological cybernetics* 43.1 (1982), pp. 59–69.

[119] Robert Kowalski and Marek Sergot. "A logic-based calculus of events". In: *Foundations of knowledge base management*. Springer, 1989, pp. 23–55.

[120] John Lafferty, Andrew McCallum, and Fernando CN Pereira. "Conditional random fields: Probabilistic models for segmenting and labeling sequence data". In: (2001).

[121] Butler Lampson. "Accountability and freedom". In: *Cambridge Computer Seminar, Cambridge, UK*. 2005, pp. 1–26.

[122] Meixing Le, Krishna Kant, and Sushil Jajodia. "Access rule consistency in cooperative data access environment". In: *8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*. IEEE. 2012, pp. 11–20.

[123] Maria Leitner and Stefanie Rinderle-Ma. "Anomaly detection and visualization in generative RBAC models". In: *Proceedings of the 19th ACM symposium on Access control models and technologies*. 2014, pp. 41–52.

[124] Mrs J Lekha, G Padmavathi, and David C Wyld. "A Comprehensive Study On Classification Of Passive Intrusion And Extrusion Detection System". In: *ICCSEA, SPPR, CSIA, WimoA-2013*. Citeseer, 2013, pp. 281–292.

[125] Ang Li et al. "Evaluating the capability and performance of access control policy verification tools". In: *MILCOM 2015-2015 IEEE Military Communications Conference*. IEEE. 2015, pp. 366–371.

[126] Weixi Li. *Automatic log analysis using machine learning: awesome automatic log analysis version 2.0*. 2013.

[127] *Logstash*. `https://www.elastic.co/logstash`.

[128] Haibing Lu, Jaideep Vaidya, and Vijayalakshmi Atluri. "Optimal boolean matrix decomposition: Application to role engineering". In: *2008 IEEE 24th International Conference on Data Engineering*. IEEE. 2008, pp. 297–306.

[129] Haibing Lu et al. "Towards user-oriented RBAC model". In: *Journal of Computer Security* 23.1 (2015), pp. 107–129.

[130] Konstantinos Makris et al. “Sparql rewriting for query mediation over mapped ontologies”. In: *Technical University of Crete* (2010).

[131] Srdjan Marinovic et al. “Rumpole: a flexible break-glass access control model”. In: *Proceedings of the 16th ACM symposium on Access control models and technologies*. 2011, pp. 73–82.

[132] Michael Mayhew et al. “Use of Machine Learning in Big Data Analytics for Insider Threat Detection”. In: ().

[133] Deborah L McGuinness, Frank Van Harmelen, et al. “OWL web ontology language overview”. In: *W3C recommendation* 10.10 (2004), p. 2004.

[134] Sheila McIlraith and Ian Horrocks. *Expressiveness question*. 2004. URL: `https://markmail.org/message/ravsqpal3p2z4hcq#query:situation%20calculus%20swrl+page:2+mid:ccq7ps5uwljixozb+state:results..`

[135] *Mediation toolkit*. `https://github.com/correndo/mediation`.

[136] Andrew Meneely, Ben Smith, and Laurie Williams. “Appendix B: iTrust electronic health care system case study”. In: *Software and Systems Traceability* (2012), p. 425.

[137] Will Mepham and Steve Gardner. “Implementing discrete event calculus with semantic web technologies”. In: *2009 Fifth International Conference on Next Generation Web Services Practices*. IEEE. 2009, pp. 90–93.

[138] Rob Miller and Murray Shanahan. “Some alternative formulations of the event calculus”. In: *Computational logic: logic programming and beyond*. Springer, 2002, pp. 452–490.

[139] Barsha Mitra et al. “Migrating from RBAC to temporal RBAC”. In: *IET Information Security* 11.5 (2017), pp. 294–300.

[140] Prasenjit Mitra et al. “Privacy-preserving semantic interoperation and access control of heterogeneous databases”. In: *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*. ACM. 2006, pp. 66–77.

[141] Decebal Mocanu, Fatih Turkmen, Antonio Liotta, et al. "Towards ABAC policy mining from logs with deep learning". In: *Proceedings of the 18th International Multiconference, ser. Intelligent Systems*. 2015.

[142] Jonathan D Moffett, Morris Sloman, and Kevin P Twidle. "Specifying discretionary access control policy for distributed systems." In: *Computer Communications* 13.9 (1990), pp. 571–580.

[143] Apurva Mohan and Douglas M Blough. "An attribute-based authorization policy framework with dynamic conflict resolution". In: *Proceedings of the 9th Symposium on Identity and Trust on the Internet*. 2010, pp. 37–50.

[144] Ian Molloy, Jorge Lobo, and Suresh Chari. "Adversaries' Holy Grail: access control analytics". In: *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*. 2011, pp. 54–61.

[145] Gary Ng. "Open vs Closed world, Rules vs Queries: Use Cases from Industry." In: *OWLED*. 2005.

[146] Qun Ni, Elisa Bertino, and Jorge Lobo. "Risk-based access control systems built on fuzzy inferences". In: *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*. 2010, pp. 250–260.

[147] P. Nimbalkar et al. "Semantic Interpretation of Structured Log Files". In: *2016 IEEE 17th International Conference on Information Reuse and Integration (IRI)*. 2016, pp. 549–555.

[148] Sami Nousiainen et al. "Anomaly detection from server log data". In: *A case study* (2009).

[149] Selmin Nurcan. "A survey on the flexibility requirements related to business processes and modeling artifacts". In: *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*. IEEE. 2008, pp. 378–378.

[150] Daniel Oberle et al. "Conceptual user tracking". In: *International Atlantic Web Intelligence Conference*. Springer. 2003, pp. 155–164.

[151] Martin J O'Connor et al. "The SWRLAPI: A Development Environment for Working with SWRL Rules." In: *OWLED*. 2008.

[152] Adam Oliner, Archana Ganapathi, and Wei Xu. "Advances and challenges in log analysis". In: *Communications of the ACM* 55.2 (2012), pp. 55–61.

[153] *OpenLink Logging Ontology.* `http://www.openlinksw.com/ontology/logging`.

[154] Keshnee Padayachee and Jan HP Eloff. "Adapting usage control as a deterrent to address the inadequacies of access controls". In: *computers & security* 28.7 (2009), pp. 536–544.

[155] Chi-Chun Pan, Prasenjit Mitra, and Peng Liu. "Semantic access control for information interoperation". In: *Proceedings of the eleventh ACM symposium on Access control models and technologies*. ACM. 2006, pp. 237–246.

[156] Symeon Papadopoulos et al. "Using event representation and semantic enrichment for managing and reviewing emergency incident logs". In: *Proceedings of the 2nd ACM international workshop on Events in multimedia*. ACM. 2010, pp. 41–46.

[157] This Paper et al. "Break-Glass – An Approach to Granting Emergency Access to Healthcare Systems". In: December (2004).

[158] Jaehong Park and Ravi Sandhu. "Towards usage control models: beyond traditional access control". In: *Proceedings of the seventh ACM symposium on Access control models and technologies*. 2002, pp. 57–64.

[159] Jaehong Park and Ravi Sandhu. "Towards Usage Control Models: Beyond Traditional Access Control". In: *Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies*. SACMAT '02. Monterey, California, USA: Association for Computing Machinery, 2002, 57–64. ISBN: 1581134967. DOI: `10.1145/507711.507722`. URL: `https://doi.org/10.1145/507711.507722`.

[160] Jaehong Park, Xinwen Zhang, and Ravi Sandhu. "Attribute mutability in usage control". In: *Research Directions in Data and Applications Security XVIII*. Springer, 2004, pp. 15–29.

[161] Anand Patwardhan et al. "Threshold-based intrusion detection in ad hoc networks and secure AODV". In: *Ad Hoc Networks* 6.4 (2008), pp. 578–599.

[162] Karl Pearson. "LIII. On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572.

[163] Frank Pfenning and Carsten Schürmann. "System description: Twelf—a meta-logical framework for deductive systems". In: *International Conference on Automated Deduction*. Springer. 1999, pp. 202–206.

[164] Javier Andres Pinto and Raymond Reiter. *Temporal reasoning in the situation calculus.* University of Toronto, 1994.

[165] Jeremy Pitt and Abe Mamdani. "A protocol-based semantics for an agent communication language". In: *IJCAI*. Vol. 99. 1999, pp. 486–491.

[166] Dean Povey. "Optimistic security: a new access control paradigm". In: *Proceedings of the 1999 workshop on New security paradigms*. 1999, pp. 40–45.

[167] *Protégé*. `https://protege.stanford.edu/products.php`.

[168] Eric Prud'hommeaux and A Seaborne. *SPARQL query language for RDF, W3C Recommendation*. Jan. 2008.

[169] Nikhil Puranik. *A specialist approach for the classification of column data*. University of Maryland, Baltimore County, 2012.

[170] Li Qin and Vijayalakshmi Atluri. "Concept-level access control for the semantic web". In: *Proceedings of the 2003 ACM workshop on XML security*. ACM. 2003, pp. 94–103.

[171] Yuzhong Qu, Xiang Zhang, and Huiying Li. "OREL: an ontology-based rights expression language". In: *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. ACM. 2004, pp. 324–325.

[172] Manfred Reichert and Peter Dadam. "ADEPT flex—supporting dynamic changes of workflows without losing control". In: *Journal of Intelligent Information Systems* 10.2 (1998), pp. 93–129.

[173] Erik Rissanen, Babak Sadighi Firozabadi, and Marek Sergot. "Towards a mechanism for discretionary overriding of access control". In: *International Workshop on Security Protocols*. Springer. 2004, pp. 312–319.

[174] *RLOG - an RDF Logging Ontology.* `https://persistence.uni-leipzig.org/nlp2rdf/ontologies/rlog/rlog.html`.

[175] Mohsen Rouached and Claude Godart. "Securing web service compositions: Formalizing authorization policies using event calculus". In: *International Conference on Service-Oriented Computing*. Springer. 2006, pp. 440–446.

[176] Farzad Salim et al. "An approach to access control under uncertainty". In: *2011 Sixth International Conference on Availability, Reliability and Security*. IEEE. 2011, pp. 1–8.

[177] Pierangela Samarati and Sabrina Capitani de Vimercati. "Access control: Policies, models, and mechanisms". In: *International School on Foundations of Security Analysis and Design*. Springer. 2000, pp. 137–196.

[178] Ravi Sandhu, Venkata Bhamidipati, and Qamar Munawer. "The ARBAC97 model for role-based administration of roles". In: *ACM Transactions on Information and System Security (TISSEC)* 2.1 (1999), pp. 105–135.

[179] Hassan Saneifar et al. "Terminology extraction from log files". In: *International Conference on Database and Expert Systems Applications*. Springer. 2009, pp. 769–776.

[180] F Scharffe. *EDOAL: expressive and declarative ontology alignment language*. 2011.

[181] Sigrid Schefer-Wenzl and Mark Strembeck. "Generic support for RBAC break-glass policies in process-aware information systems". In: *Proceedings of the 28th annual ACM symposium on applied computing*. 2013, pp. 1441–1446.

[182] *Security recommendations for implementation a logging system*. `https://www.ssi.gouv.fr/uploads/IMG/pdf/NP_Journalisation_NoteTech.pdf`.

[183] Fei Sha and Fernando Pereira. "Shallow parsing with conditional random fields". In: *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*. 2003, pp. 213–220.

[184] Shaul Shalvi, Ori Eldar, and Yoella Bereby-Meyer. "Honesty requires time (and lack of justifications)". In: *Psychological science* 23.10 (2012), pp. 1264–1270.

[185] Murray Shanahan. "The event calculus explained". In: *Artificial intelligence today*. Springer, 1999, pp. 409–430.

[186] Nitin Kumar Sharma and Anupam Joshi. "Representing attribute based access control policies in owl". In: *2016 IEEE Tenth International Conference on Semantic Computing (ICSC)*. IEEE. 2016, pp. 333–336.

[187] Cheng-chun Shu, Erica Y Yang, and Alvaro E Arenas. "Detecting conflicts in ABAC policies with rule-reduction and binary-search techniques". In: *2009 IEEE International Symposium on Policies for Distributed Systems and Networks*. IEEE. 2009, pp. 182–185.

[188] Tuomo Sipola, Antti Juvonen, and Joel Lehtonen. "Anomaly detection from network logs using diffusion maps". In: *Engineering Applications of Neural Networks*. Springer, 2011, pp. 172–181.

[189] Waleed W Smari, Patrice Clemente, and Jean-Francois Lalande. "An extended attribute based access control model with trust and privacy: Application to a collaborative crisis management system". In: *Future Generation Computer Systems* 31 (2014), pp. 147–168.

[190] Clóvis Eduardo de Souza Nascimento et al. "OntoLog: Using Web Semantic and Ontology for Security Log Analysis". In: *ICSEA 2011*. 2011.

[191] *SparqlToXQuery*. `https://sourceforge.net/projects/sparqltoxquery/`.

[192] *Sysmon logs*. `https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon`.

[193] *The Syslog Protocol*. `https://www.ietf.org/rfc/rfc5424.txt`.

[194] Chrisa Tsinaraki and Stavros Christodoulakis. "Interoperability of XML schema applications with OWL domain knowledge and semantic web tools". In: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer. 2007, pp. 850–869.

[195] *Types of Access Control*. `http://cisspstudy.blogspot.com/2007/05/types-of-access-control.html`.

[196] Andrzej Uszok, Jeffrey M Bradshaw, and Renia Jeffers. "Kaos: A policy and domain services framework for grid computing and semantic web services". In: *International Conference on Trust Management*. Springer. 2004, pp. 16–26.

[197] Jaideep Vaidya, Vijayalakshmi Atluri, and Qi Guo. "The role mining problem: finding a minimal descriptive set of roles". In: *Proceedings of the 12th ACM symposium on Access control models and technologies*. 2007, pp. 175–184.

[198] Wil Van Der Aalst. "Process mining". In: *Communications of the ACM* 55.8 (2012), pp. 76–83.

[199] Boudewijn F Van Dongen et al. "The ProM framework: A new era in process mining tool support". In: *International conference on application and theory of petri nets*. Springer. 2005, pp. 444–454.

[200] Anthony Vance, Paul Benjamin Lowry, and Dennis Eggett. "Using accountability to reduce access policy violations in information systems". In: *Journal of Management Information Systems* 29.4 (2013), pp. 263–290.

[201] Sabrina De Capitani di Vimercati and Pierangela Samarati. "Authorization specification and enforcement in federated database systems". In: *Journal of Computer Security* 5.2 (1997), pp. 155–188.

[202] Paul Voigt and Axel Von dem Bussche. "The eu general data protection regulation (gdpr)". In: *A Practical Guide, 1st Ed., Cham: Springer International Publishing* (2017).

[203] Jacques Wainer. "Anomaly Detection using Process Mining". In: (), pp. 1–13.

[204] Jacques Wainer, Paulo Barthelmess, and Akhil Kumar. "W-RBAC—A workflow security model incorporating controlled overriding of constraints". In: *International Journal of Cooperative Information Systems* 12.04 (2003), pp. 455–485.

[205] M. Wang, L. Xu, and L. Guo. "Anomaly Detection of System Logs Based on Natural Language Processing and Deep Learning". In: *2018 4th International Conference on Frontiers of Signal Processing (ICFSP)*. 2018, pp. 140–144.

[206] Yigong Wang et al. "Conflicts analysis and resolution for access control policies". In: *2010 IEEE International Conference on Information Theory and Information Security*. IEEE. 2010, pp. 264–267.

[207] AJMM Weijters, Wil MP van Der Aalst, and AK Alves De Medeiros. "Process mining with the heuristics miner-algorithm". In: *Technische Universiteit Eindhoven, Tech. Rep. WP* 166 (2006), pp. 1–34.

[208] Peter C Weinstein and William P Birmingham. *Agent communication with differentiated ontologies: eight new measures of description compatibility*. Tech. rep. Michigan Univ Ann Arbor Dept Of Electrical Engineering And Computer Science, 1999.

[209] Gio Wiederhold. "Mediators in the architecture of future information systems". In: *Computer* 25.3 (1992), pp. 38–49.

[210] Michael Wooldridge and Nicholas R Jennings. "Intelligent agents: Theory and practice". In: *The knowledge engineering review* 10.2 (1995), pp. 115–152.

[211] Wei Xu et al. "Detecting large-scale system problems by mining console logs". In: *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. 2009, pp. 117–132.

[212] Zhongyuan Xu and Scott D Stoller. "Mining attribute-based access control policies from logs". In: *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer. 2014, pp. 276–291.

[213] Zhongyuan Xu and Scott D Stoller. "Mining attribute-based access control policies from RBAC policies". In: *2013 10th International Conference and Expo on Emerging Technologies for a Smarter World (CEWIT)*. IEEE. 2013, pp. 1–6.

[214] Guangsen Zhang and Manish Parashar. "Context-aware dynamic access control for pervasive applications". In: *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference*. 2004, pp. 21–30.

UNIVERSITE BRETAGNE LOIRE MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Titre : Analyse a posteriori des logs et détection de violation des règles de sécurité

Mot clés : Contrôle d'accès, Analyse des logs, Vérification temporelle, Violations, Sanctions.

Résumé : Dans certains environnements sensibles, tels que le domaine de la santé, où les utilisateurs sont généralement de confiance et où des évènements particuliers peuvent se produire, comme les situations d'urgence, les contrôles de sécurité mis en place dans les systèmes d'information correspondants ne doivent pa bloquer certaines décisions et actions des utilisateurs. Cela pourrait avoir des conséquences graves. En revanche, il est important de pouvoir identifier et tracer ces actions et ces décisions afin de détecter d'éventuelles violations de la politique de sécurité mise en place et fixer les responsibilités. Ces fonctionnalités sont assurées par le contrôle d'accès a posteriori qui se base un mécanisme de monitoring à partir des logs.
Dans la littérature, ce type de contrôle de sécurité a été divisé en trois étapes qui sont : le traitement des logs, l'analyse des logs, et l'imputabilité.
Dans cette thèse, nous couvrons ces trois domaines du contrôle d'accès a posteriori en apportant de nouvelles solutions, et nous introduisons des nouveaux aspects qui n'avaient pas été abordés auparavant.

Title: A posteriori log analysis and security rules violation detection

Keywords: Access Control, Log Analysis, Temporal verification, violations, sanctions.

Abstract: In certain sensitive environments, such as the healthcare domain, where users are generally trusted and where particular events may occur, such as emergencies, the implemented security controls in the corresponding information systems should not block certain decisions and actions of users. This could have serious consequences. Indeed, it is important to be able to identify and trace these actions and decisions in order to detect possible violations of the security policy put in place and fix responsibilities. These functions are ensured by the a posteriori access control that lies on a monitoring mechanism based on logs.
In the literature, this type of access control has been divided into three stages: log processing, log analysis, and accountability.
In this thesis, we cover these three areas of the a posteriori access control by providing new solutions, and we introduce new aspects that have not been addressed before.