KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Application-level authorization for multi-organizational software

Jasper Bogaerts

Supervisor:
Prof. dr. ir. W. Joosen
Dr. B. Lagaisse, co-supervisor

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

August 2018

Application-level authorization for multi-organizational software

Jasper BOGAERTS

Examination committee:
Prof. dr. ir. O. Van der Biest, chair
Prof. dr. ir. W. Joosen, supervisor
Dr. B. Lagaisse, co-supervisor
Prof. dr. ir. Y. Berbers
Prof. dr. ir. E. Steegmans
Dr. S. Petkova-Nikova
Prof. dr. ir. B. Preneel
Prof. dr. ir. F. De Turck
(University of Ghent)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

August 2018

Uitgegeven in eigen beheer, Jasper Bogaerts, Celestijnenlaan 200A box 2402, B-3001 Leuven (Belgium)

Preface

When I first started working on the research that is summarized in this thesis, I could not have imagined the incredibly fruitful journey I would set upon. Now, years later, I can look back at a time full of compelling challenges.

First of all, I would like to thank my supervisor, Wouter Joosen, for giving me the opportunity to have this unique experience. Wouter offered me the time and freedom to pursue my interests, and provided valuable advice that helped me broaden my horizon. This has not only changed my perspective on computer science, but also on career and life goals. The tireless effort he puts in the development of DistriNet has really made it a place you can always feel at home at.

Secondly, I want to thank Bert Lagaisse. Bert, as my coach and co-supervisor, was always there to help me when I was facing problems. He always took the time to listen to me and helped me resolve any issue I had along the way. I can only have admiration for the patience that he has shown in coaching me and giving me invaluable advice on all aspects of my work.

Another thanks goes to the members of my jury, who provided interesting novel perspectives on my work. Omer Van der Biest, Yolande Berbers, Eric Steegmans, Svetla Petkova-Nikova, Bart Preneel, and Filip De Turck, thank you for the time and effort you took in proof-reading my text, and the thought-provoking questions and valued feedback that helped me improve my work.

Next, I would like to give a big thanks to the colleagues I had the pleasure to work with over the years. It has been an honor and a privilege to work with such talented and remarkable people. In particular, I want to issue a special thanks to Maarten Decat. Our collaboration has been an exceptionally enlightening experience to me, which taught me valuable things that I will carry with me a long time. To my (former) office co-workers, Bart, Wouter DB, Stefan, Jan, Ansar, Tim, Jasper M, Emad, Tung, Mario, Jorik, Arnaud: it was always a pleasure coming in to work in such a nice and relaxed atmosphere. My gratitude

also goes out to all the members of Lantam. You always gave me interesting feedback on my work, and our meetings kindled inspiring discussions on a broad range of research domains. Also, I want to thank the colleagues I collaborated with on the various courses that I helped tutor in the last years, for their helpful support and advice. Another thanks goes out to the administrative and technical staff for making our lives easier, and contributing to the atmosphere at the department.

A special mention goes out to my parents, friends, family, and family in law for always supporting and helping me through the hard moments during this experience. You always lend a listening ear and helped me look at things from a different perspective.

Last but not least, I would like to thank my love, Chlara, for everything she has done for me. Chlara, you have always been there for me, giving me advice on crucial moments, and helping me cope with the hardships that I have faced over the last years. I could not have done this without you, and can only hope to be able to provide the same support to you in your time of need.

– Jasper Bogaerts, August 2018

Abstract

This dissertation addresses the problem of enforcing multi-organizational authorization policies for collaborative software applications, and presents a security middleware that enables fine-grained run-time customization of such authorization policies.

Support for multiple organizations is becoming an increasingly important concern in contemporary software systems and services. This need appears in a wide range of application domains, ranging from software delivery models such as Software as a Service (SaaS) to applications that require collaboration between independent parties. Additionally, a growing concern is the ability to tailor the application to the business processes and organizational needs of its clients. Software security technologies need to follow suit and pursue a general approach that satisfies these requirements.

In terms of access control, and application-level authorization in particular, however, several challenges remain. First, software that aspires to accommodate several organizations simultaneously requires a systematic approach that supports run-time customization and enforcement of authorization policies of these organizations. Second, authorization enforcement demands multi-tier support that comprises data-focused operations such as search, especially in support of novel, more expressive policy languages. Third, the expressiveness of policy languages is central to the adoption of multi-organizational authorization approaches. These languages need to support both intra- and inter-organizational associations and their corresponding properties as part of their capabilities to specify authorization constraints.

The goal of this dissertation is to address these challenges in order to support multi-organizational authorization. To this end, we provide three contributions that focus on both management and enforcement issues. First, we present Amusa, an authorization middleware for multi-organizational, collaborative applications that enables self-management of policies by multiple individual

organizations. AMUSA supports both specification and enforcement of multi-organizational authorization policies that can be customized at run-time. Second, we present SEQUOIA, an authorization middleware for enforcing attribute-based policies on search and aggregation operations in data-driven applications in a scalable manner. The SEQUOIA middleware supports the capabilities supported by AMUSA beyond the realm of control-focused operations. Third, we present entity-based access control, an authorization system that includes a policy model, language, and enforcement system that increases expressiveness of the authorization policies. As such, this system enables these policies to reason in a more seamless way about the application domain on which they apply. This increases expressiveness for domain experts.

These contributions are inspired by four industry case studies in the domains of security monitoring and compliance, document processing, electronic health record management, and workforce management. Additionally, we have performed a thorough evaluation of our contributions in terms of integration effort and performance impact based on extensive prototypes. Both validations and evaluation demonstrate that our contributions effectively address the aforementioned challenges, and therefore significantly improve the support for multi-organizational authorization.

Beknopte samenvatting

Deze thesis behandelt het probleem van het afdwingen van multi-organisationele toegangsregels voor software toepassingen voor samenwerkende organisaties. Het introduceert ook een beveiligingsmiddleware die fijnmazige, run-time aanpassingen van deze toegangsregels ondersteunt.

Huidige software toepassingen hebben steeds meer nood aan ondersteuning voor meerdere organisaties die tegelijkertijd hun taken op de software uitvoeren. Deze nood is aanwezig in een brede waaier van applicatiedomeinen, zoals onder andere in Software as a Service (SaaS) toepassingen. Er is bovendien een toenemende vraag naar de mogelijkheid om toepassingen te configureren op basis van bedrijfsprocessen en organisationele noden van de klanten. Softwarebeveiligingstechnologieën moeten hierin volgen en een aanpak nastreven die aan deze vereisten voldoet.

Voor toegangscontrole bestaan er in deze context echter nog enkele uitdagingen. Ten eerste is er een systematische aanpak nodig ter ondersteuning van run-time aanpassing en bekrachtiging van toegangsregels binnen toepassingen die meerdere organisaties tegelijkertijd ondersteunen. Ten tweede moet de bekrachtiging van deze toegangsregels op meerdere niveauś worden ondersteund, onder meer ook voor data–intensieve handelingen zoals zoekopdrachten, voornamelijk in de context van nieuwere, expressieve specificatietalen die de toegang omschrijven. Ten derde staat de expressiviteit van deze specificatietalen centraal in de adoptie van technologieën voor multi-organisationele toegangscontrole. Deze talen moeten daarom associaties zowel binnen een organisatie als tussen organisaties ondersteunen, en moeten de kenmerken van deze organisaties mee laten kunnen bepalen of toegang geoorloofd is.

Het doel van deze thesis is om deze uitdagingen aan te pakken teneinde de multi-organisationele toegangscontrole beter te ondersteunen. Om dit te bereiken introduceren we drie contributies die de nadruk leggen op zowel beheers- als bekrachtigingsproblemen van toegangscontrole. Ten eerste introduceren

we Amusa, een middleware ter ondersteuning van multi-organisationele toegangscontrole die het zelfbeheer van toegangsregels door verschillende onafhankelijke partijen toelaat. Amusa ondersteunt zowel de specificatie als bekrachtiging van toegangsregels die kunnen worden aangepast *at run-time*. Ten tweede introduceren we Sequoia, een middleware die op een schaalbare manier de bekrachtiging van attribuut-gebaseerde toegangsregels ondersteund voor zoekopdrachten en handelingen voor data–analyse. Zo verbreedt Sequoia de ondersteuning van Amusa tot data–intensieve handelingen. Ten derde introduceren we entiteits-gebaseerde toegangscontrole door middel van een specificatiemodel, specificatietaal, en toegangscontrolesysteem dat de expressiviteit van toegangscontroleregels verbreedt. Dit systeem ondersteunt specificatie van toegangsregels op een meer flexibele manier die dichter aanleunt met het toepassingsdomein waarvoor ze worden gespecificeerd, wat de expressiviteit verhoogt voor domein-experten.

Deze contributies werden geïnspireerd door vier gevalanalyses uit de industrie, in de domeinen van beveiligingstoezicht, documentbeheer, e-health, en personeelsbeheer. Daarenboven voerden we een uitgebreide performantieanalyse door, gebaseerd op omvangrijke prototypes van onze contributies. Zowel de validatie als de evaluatie van onze contributies tonen aan dat ze de bovenvermelde uitdagingen aanpakken, en daardoor een aanzienlijke bijdrage leveren aan de ondersteuning van multi-organisationele toegangscontrole.

Abbreviations

MAC	Mandatory access control
DAC	Discretionary Access Control
SoD	Separation of Duty
CRUD	Create, Read, Update, Delete
IBAC	Identity-Based Access Control
LBAC	Lattice-Based Access Control
RBAC	Role-Based Access Control
ABAC	Attribute-Based Access Control
ReBAC	Relationship-Based Access Control
EBAC	Entity-Based Access Control
PBAC	Policy-Based Access Control
PEP	Policy Enforcement Point
PAP	Policy Administration Point
PIP	Policy Information Point
PDP	Policy Decision Point
CH	Context Handler
ObS	Obligation Service
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service
PO	Permit Overrides
DO	Deny Overrides
FA	First Applicable

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since the advent of computers, software has gradually taken a central role in our world. Nowadays, enterprise software is ubiquitous, facilitating nearly every business responsibility and optimizing business processes both internally and between organizations. In organizations, applications now assume a wide range of responsibilities such as work flow planning, document management, enterprise resource planning, and contract negotiation. As such, the importance of software in business is constantly increasing, as applications are continuously taking a more central position in business practices.

Because of the key importance of software in businesses, being able to properly secure these systems has become paramount. For example, security measures must be taken to prevent malicious persons from discovering the conditions of contracts they are not involved in. As attention to securing software systems has increased over the last decades, the computer security industry has grown into the multi-billion dollar industry it is today [164].

Software security comprises a large number of aspects, including for example encryption to client-side web security. In this thesis, we focus on access control, and *application-level authorization* in particular. Authorization restricts the actions that users can perform on computer resources in a certain context. For example, while a director should be permitted to confirm a negotiated contract for the organization, not every employee is typically entitled to do so unless explicitly specified by the organization's authorization policy.

Software systems are becoming increasingly complex, putting some important challenges to the forefront. This includes challenges to improve fault tolerance, system scalability, and coping with variability of functionality. Another

important requirement for contemporary software systems is the ability for various parties (e.g., end-users, enterprise customers, third parties) to *customize* a wide range of aspects in order to satisfy their personal or organizational preferences. For access control, however, a comprehensive, systematic support for the security management of computer resources in a *multi-organizational* setting is still lacking. Multi-organizational applications regard a set of software resources that is partially shared among a subset of parties that aspire to customize access restrictions for the resources they own, thereby complicating authorization management and enforcement. As a consequence, organizations can insufficiently tailor access control properties to their security needs, and need to make compromises that deteriorate the overall security of the application they manage. Security misconfiguration and access control issues have also been identified as a top priority by several prominent security surveys [81, 116, 230]. Nonetheless, proper authorization measures are imperative to multi-organizational applications such as contract negotiation software, as they require compliance to both organizational as well as local regulations, and may involve sharing of resources on which authorization must be enforced.

This thesis addresses this challenge by focusing on application-level security middleware that improve support for authorization management and enforcement for such software applications. In particular, we improve support for multi-organizational applications through (i) a security middleware that supports customization in collaborative applications, (ii) a data access middleware that extends this customization support for data-focused operations, and (iii) an access control model and policy evaluation engine that improves authorization policy specification granularity for multi-organizational applications.

This chapter provides an introduction to this work. First, it briefly introduces access control and the concept of multi-organizational applications. Next, the chapter outlines some key challenges with regard to multi-organizational authorization. Based on these challenges, it stipulates the goals this thesis has aimed to achieve, and describes the research approach that was taken to achieve these goals. Finally, this chapter summarizes the contributions and outlines the remainder of the thesis.

1.1 Access control and multi-organizational applications

Access control is a security discipline that covers authentication, authorization and auditing of subjects (e.g., users) that must be restricted from performing

Table 1.1 – Access control spans multiple domains on multiple abstraction levels. For each domain (authentication, authorization, audit), it regards management and enforcement. This table lists technologies or research prototypes within each realm and solidifies our focus.

Focus	Authentication	Authorization	Audit
Operating System	e.g., LDAP	e.g., SELinux [159]	e.g., auditctl [106]
Network	e.g., SSH	e.g., Firewalls [153]	e.g., IDS [14]
Database	e.g., Multi-tier authentication [41]	e.g., FGAC [195]	e.g., [126]
Application	e.g., Shibboleth [123]	**Our focus**	e.g., TaintDroid [87]

actions on a software system. Authentication encompasses the verification process of the identity of a subject. Authorization, on the other hand, comprises the restriction of the actions that a subject can perform on objects (i.e., computer resources) in a certain context. Audit consists of collecting and analyzing logged access control data in order to determine whether security violations have occurred.

Research in access control entails a wide range of domains. This is illustrated in Table 1.1. Access control research spans from operating system access control (e.g., SELinux [159]), network security (e.g., firewalls [153]), database access control (e.g., [26]), to application-level access control. Other research focuses on more fundamental concerns, such as access control logic [1, 134], policy analysis [218, 228], and access control models [90, 204] and migration between policies based on these models [137, 237].

Our primary focus in this dissertation is on *application-level authorization* for multi-organizational software applications. For this aspect, we focus on both management and enforcement approaches that facilitate security administrators to specify and reinforce authorization policies in a multi-organizational context.

We achieve this through security *middleware*. A middleware abstracts from complex interactions with other software components, and serves as a layer from which the application component(s) communicate over the network, with the operating system, and with database systems to ensure data persistence. As such, it can also perform additional functionality, such as security, analytics, or performance optimization.

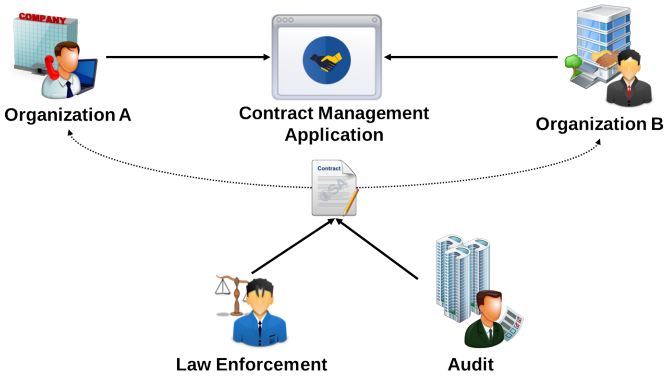


Figure 1.1 – Applications facilitating the management and signing of legal contracts typically need to provide access to multiple parties, to a possibly shared set of resources.

Multi-organizational applications

This dissertation concentrates on multi-organizational software applications. A *multi-organizational application* provides a common interface to multiple organizations and/or types of users that use the software to create, manage and operate on a set of data resources that is typically (but not exclusively) at least partially shared among subsets of organizations. Such organizations generally require customization of their security configuration, and may have conflicting interests with regard to the shared data resources. In essence, multi-organizational applications share both software, infrastructure, and a subset of resources that involved organizations collaborate on among each other.

For example, consider a software platform that facilitates the setting up, negotiation, management and registration of legal contracts [67, 68]. This is also depicted in Figure 1.1. Such applications require cooperation of multiple organizations that negotiate the shared contract, but could also involve support for fiduciary parties, auditing organizations, and government entities that must have access to a restricted set of contracts. Contract management platforms face many technical challenges, ranging from service scalability optimizations, robustness measures, to providing accurate analysis concerning various properties of the negotiation process and the resulting contracts. Such negotiation processes produce artifacts that correspond to a specific contract and require access restrictions. Contract negotiations adhere to an established workflow that can be customized in order to ensure that they comply to organizational, sectoral, and regional regulations. For example, negotiation of a certain type of legal contract may require that particular steps in the process must be reviewable by an external auditor, or otherwise approved by a

governmental agency, which could be customized according to the region for which the contract applies.

Customization support is also typically expected for security configurations. For various reasons, not in the least for purposes of retaining competitive advantages, this application must be able to enforce both inter- as well as intra-organizational access control restrictions in order to achieve confidentiality of the managed contracts. Organizations are likely only permitted access to consult any contracts they participate in. Inside the organization this may be further restricted to specific persons who are involved with the contract negotiation. Such restrictions must be manageable by security administrators associated with that organization itself. The software platform serves as a trusted entity that is responsible for enforcing these restrictions over all operations that can be performed on its resources. The platform also may specify its own authorization constraints according to the application provider's security preferences and in order to ensure regulatory compliance.

Multi-organizational applications are becoming increasingly prevalent. This trend is amplified by the increased adoption of Software as a Service applications as a part of the IT infrastructure in various organizations [66, 140], which is a subset of multi-organizational applications. Such applications introduce particular requirements with regard to access control specification and enforcement. In particular, they require *run-time customization* of access control configurations, as the dynamicity of the platform imposes that specific access restrictions must be put in place that are usually not entirely clear at design nor deployment time. Moreover, since multiple parties may be involved in the operations performed on the computer resources, the application should be capable of mediating between and prioritizing over (possibly conflicting) access control configurations from various parties to a shared resource.

It is impossible for a software vendor to know what access control configurations should be enforced at the time that an application is deployed. These restrictions are influenced by local regulations, internal organizational policies, and even the nature of the individual resources managed by the platform. For example, an organization may constrain access to any artifacts of a contract negotiated with another organization to only managers who are responsible for perpetuating services that are offered by that other organization. Access in such case could also be limited to working hours. Another organization may only allow access to specific negotiation teams that are assigned the enterprise customers they are doing the mediation for. Yet another organization may be imposed to audit the contracts, and could be compelled by regulation to take measures against conflicts of interest, therefore barring their employees to view certain contracts of one organization after consulting others [42].

Such access control configurations are subject to changes, as regulations and organizational policies may evolve over time. Moreover, the multi-organizational applications should support access control configurations to align with organizational preferences as much as possible, without introducing prohibitive enforcement overhead.

1.2 Multi-organizational authorization challenges

The nature of multi-organizational applications is to accommodate requests from multiple independent parties (e.g., users, enterprise customers, and third parties such as government agencies). This poses challenges, not in the least for security. While many security techniques can be applied modularly (such as communication over secure channels, session hijacking countermeasures for web-based applications and code-injection prevention techniques), access control, and application-level authorization in particular, poses specific challenges. This section elaborates on these challenges with regard to authorization for multi-organizational applications.

The main focus for this dissertation is the expression, management, and comprehensive enforcement of customizable authorization privileges in a multi-organizational setting. The authorization management aims to support variability of access control configuration for each involved organization, as is also a common requirement from a functionality perspective in Software as a Service applications [221]. In this context, we identify six challenges. These challenges necessitate fundamental approaches that improve authorization management and enforcement in a multi-organizational setting.

Organization-centric expressiveness. Multi-organizational applications should enable security administrators to specify authorization constraints that align *as close as possible* to both the organizational structure and the desires of the organization on behalf of which they perform administration. This imposes support for variability of authorization configurations, and requires support for *expressiveness* of those authorization constraints, without introducing prohibitive administrative overhead. Since these authorization constraints deal with resources in multi-organizational applications, the expressiveness should be *organization-centric*. This enables associations between organizations and their employees as well as the properties they hold to have an impact on authorization decisions. For example, an authorization constraint should support the expression that any organization can access all artifacts related to a contract negotiation they are involved with, unless that artifact is classified as holding secret information.

Self-management. Organizations should be enabled to specify their own authorization constraints. This concept, referred to as *self-management*, increases the administrative scalability of multi-organizational applications with regard to access control. For instance, if an application serves many organizations, it would otherwise risk spending considerable effort in managing authorization constraints for them, thereby becoming a significant cost factor. Self-management alleviates this through supporting organizations to align their authorization constraints closer with the organizational structure as it removes the overhead of communicating preferences to the application provider. In addition, self-management enforces a form of *separation of concerns* [82, 182] in which security administrators associated with the organization are responsible for specifying the access control configuration instead of an employee of the application provider. This avoids the latter from requiring to have a deep understanding of the organizational structure, processes, and preferences of the relevant enterprise customer. Separation of concerns has been argued to improve security [74] and reduces costs for the application provider associated with management.

Comprehensive enforcement. Security administrators should be able to specify authorization constraints involving any operation that can be performed by the application. This not only includes operations associated with creating, reading, or modifying resources, or domain-specific operations (e.g., signing a contract), but also *data-focused* operations that concern searching of resources or performing data aggregation (e.g., categorization of contracts). For data-focused operations, authorization should be sufficiently fine-grained as to include only elements in the result if the acting user has the appropriate privileges to it. For example, an employee searching for contracts that satisfy a certain set of constraints should only be presented the contracts he/she is entitled to see.

Isolation. When operating an application that accommodates multiple organizations, access of an end-user should generally be constrained to resources of the organization that user is affiliated with. For example, an organization should typically not have access to the artifacts of contracts associated with another organization because this could involve data that could give the former organization a competitive advantage or lead to a conflict of interest.

Sharing. Although isolation should be enforced in common situations, multi-organizational applications may also support the *sharing* of resources among different organizations. For example, contract negotiations require resources involved with the negotiation to be shared among the participating organizations. This implies that methods should be in place that enable exceptions to the authorization constraints stemming from the *isolation* constraint. However, such exceptions should only apply to the involved organizations, and in case they explicitly indicate their desire to do so.

Multi-layered management. Authorization should be managed from a top-down manner in order to retain a clear and scalable security administration. This entails that generally applicable authorization constraints such as *isolation* should be encapsulated, and that the application provider should be able to specify its own authorization constraints that reason about the enterprise customers and third parties. On the other hand, the *self-management* challenge implies that organizations can reason about access attempts on resources that they own. Moreover, larger organizations could delegate administrative responsibilities to internal divisions and departments that perform authorization management for a subset of employees. For example, in order to comply to regulatory commitments, the application provider may restrict certain types of contract negotiations between competitors. Organizations could entrust security administrators of sales and legal departments to perform a portion of the authorization management.

Non-functional requirements

Besides these challenges, authorization in multi-organizational applications should also generally prioritize four *cross-cutting* non-functional requirements: modifiability, centralization, reusability, and performance. These requirements should be focused on for all the aforementioned challenges, and thus compel significant attention in the contributions addressing them.

Modifiability. Multi-organizational applications should support access control configurations to be modified *at run-time*, as opposed to requiring the configuration to remain fixed after the software development phase or after deployment to the server. Because the authorization constraints of organizations may change and regulating guidelines evolve over time, there should be support to accommodate this issue at run-time. Moreover, it is sometimes impossible to know in advance which organizations will be served by the application when it is deployed. For example, an external audit organization may be called in to look over the negotiated contracts at a time that the application is already being used. Run-time modifiability is a requirement that emanates from the need for near-continuous up-time in shared software systems such as multi-organizational applications. Because organizations rely on the availability of the services from the application to complement their own computer infrastructure, the application provider is compelled to ensure that this application remains continuously operable.

Centralization. Our focus in this thesis is on applications that are deployed on a central computer infrastructure. Hence, authorization management and enforcement is provided through a *central authority* that supports multiple

administrative domains. Centralization reduces the management costs and efforts, and facilitates monitoring of systems. However, this requires the organizations to trust the platform for the enforcement of their authorization configuration.

Reusability. A multi-organizational application might be able to address the above requirements in a domain-specific way. However, we would instead benefit dramatically from these concerns being addressed in a generic and systematic manner by means of a general framework. Making the approach *reusable* would allow software developers to focus on the core functionality of their application instead of having to concentrate on authorization. This also supports the concept of separation of concerns [74], as reusable and encapsulated access control functionality enables both software developers and security experts to focus on their core responsibilities.

Performance. As important as software security may have become today, it is not the core functionality of multi-organizational applications. Consequently, security measures should not incur a considerable overhead on the processing times that requests to the application generate. As such, performance overhead caused by access control should be minimized as much as possible.

1.3 Goals and research approach

This dissertation has aspired to address these requirements, putting an emphasis on the organization-centric expression, management, and comprehensive enforcement of authorization privileges. In order to do so, it aimed to analyze, incept, develop and evaluate approaches in support of authorization in multi-organizational applications. The contributions that underpin this purpose focus both on authorization management and enforcement techniques to achieve their goals. As there is a large emphasis on reusability of these approaches, the contributions aim to encapsulate the proposed approaches in reusable security middleware. These middleware platforms are intended to pertain to multiple application domains.

Research approach. In order to achieve the aforementioned goal, the research approach for this thesis has concentrated around four aspects.

1. **Driven by industry case-studies.** The work was driven by industry cases in the domains of workforce planning [76], electronic document processing [75], security monitoring and compliance services, and electronic health record management. These cases drove the inception

of authorization methods, as they provided real-world requirements and challenges that have to be addressed in order to come to a general approach.

2. **Leveraging proven technologies.** This dissertation uses several seminal technologies as a basis to advance the state-of-the-art in authorization. These technologies include (i) policy-based access control [200], (ii) tree-structured policy components [172], (iii) a many-valued logic system, and (iv) attribute-based access control [120]. They are the basic building blocks for the methods proposed in this thesis.

3. **Validated through prototypes.** Each of the contributions was validated by a prototype implementation that ensured feasibility of the approach and provided valuable insights into its complexities and pitfalls. As per the requirements of the previous section, each prototype was incepted into a reusable middleware that was validated against multiple industry cases.

4. **Supported by extensive evaluation.** Each prototype was subjected to an extensive evaluation that assessed its performance overhead and the security properties of the method on which it was based. The results of these evaluations are presented throughout the thesis alongside the contributions that they assess.

All contributions presented in this thesis adopted this research approach. The next chapter introduces the industry case studies and discusses supporting technologies that were leveraged in this dissertation in more depth. Discussion, validation and evaluation of the contributions is deferred to subsequent chapters.

1.4 Contributions and outline of the thesis

This dissertation provides three major self-contained contributions that support management and enforcement of authorization in multi-organizational applications.

First, Chapter 2 provides a background for the thesis, introducing industry case studies and elaborating on key access control concepts. The chapter also discusses related research domains and discusses the core technologies that were leveraged by the contributions. Also, it briefly elaborates on the contributions themselves.

Next, Chapters 3–5 present the individual contributions, that are also outlined further in this section, in greater detail. Each of the contributions is described in-depth in the corresponding chapter, including problem elaboration, evaluation,

and further discussion. All of the contributions address a subset of the challenges that were introduced in Section 1.2.

Chapter 6, finally, summarizes the outcomes of our work and discusses future directions for access control.

Contributions

The three contributions regard both authorization management and enforcement to some extent. They all address a subset of the challenges discussed in Section 1.2. In particular, modifiability, centralization, reusability, and performance were persevered. The other challenges were all addressed to some extent in the contributions, putting an emphasis on authorization management, comprehensive enforcement, and organization-centric expressiveness, respectively. The remainder of this section provides a general outline of the contributions.

Contribution 1. Amusa: Run-time customization management and enforcement middleware for authorization in multi-organizational, collaborative applications.

This contribution proposes Amusa, a security middleware that supports both management and enforcement for authorization in multi-organizational applications.

The Amusa middleware facilitates development and administration of such applications by supporting flexible, declarative authorization management. This management enables run-time customization of authorization constraints, supporting self-management and multi-layer management to scale administrative responsibilities. The middleware empowers the sharing of resources, but also enforces isolation of the involved organizations by default. As such, it enables respective organizations to specify their own authorization preferences, and provides a general framework that outsources authorization management logic from collaborative applications.

Besides a management framework, Amusa also comes with an enforcement infrastructure that provides authorization decisions to the application at a low performance overhead through configurable performance tactics. Chapter 3 describes this contribution in greater detail.

Contribution 2. Sequoia: a scalable authorization middleware for search and aggregation operations in data-driven, multi-organizational applications.

Amusa focused primarily on authorization management, and provided an enforcement infrastructure that is directed to authorization on control-focused operations (e.g., signing a contract). Sequoia extends these concepts for *data-driven* applications by providing a data access middleware that enforces the aforementioned authorization constraints on data-focused operations such as search and data aggregation.

Through query rewriting based on dynamic run-time conditions, the security middleware effectively restricts these data-focused operations so that users only obtain data elements (i.e., resources) as part of these operations to which they are privileged. The rewriting approach supports technologies such as attribute-based access control, many-valued decision sets and policy trees through transformation to boolean expressions.

Sequoia provides support for both SQL and NoSQL database technologies and yields a low performance overhead. The transformation techniques that support the core technologies are formally verified and the approach was validated through a prototype. As such, Sequoia provides support for comprehensive enforcement of authorization constraints in multi-organizational applications. Chapter 4 describes this contribution in depth and presents an extensive performance evaluation of the validating prototype.

Contribution 3. Entity-Based Access Control: an entity-based authorization policy language and evaluation system that increases expressiveness of policies for multi-organizational applications.

This contribution extends the state-of-the-art authorization systems through a policy model and enforcement system that regards entities as first-class citizens in policies and considers attributes and relationships to evaluate these policies.

Previous research has focused on Role-Based Access Control (RBAC, [90]) to accommodate multiple organizations in an application [143]. This has been extended with novel access control models such as Attribute-Based Access Control (ABAC, [119]) for collaborative applications [223] and Relationship-Based Access Control (ReBAC, [46]), primarily in the context of social networks [59]. While RBAC models suffer from management issues, among others when the specification ownership-based authorization constraints is required, ABAC supports natural specification of a large variety of constraints but does not support natural specification of relationships. ReBAC, on the other hand, does support a natural specification of relationships but constraints on other properties cannot be specified naturally. Our case studies show that both

properties and relationships are necessary to naturally express authorization constraints in multi-organizational applications.

Through Entity-Based Access Control (EBAC), we regard entities as first-class citizens in authorization policies and take into account both attributes and relationships to evaluate these policies. As such, we provide a policy language, and a corresponding policy evaluation engine, that enables security experts to specify policies that more closely align to the multi-organizational application domain. As such, EBAC enables organization-centric expressiveness of authorization constraints, thus supporting opportunities for authorization management in multi-organizational applications. This contribution is described in detail in Chapter 5.

Chapter 2

Case Studies, Motivation and Background

This chapter provides a deeper inspection into the background in the domain of authorization in multi-organizational applications. First, we present several motivating case studies that illustrate and inspire important challenges to the domain. Second, we provide a general overview of access control in general and research domains that are related to multi-organizational applications. Third, we discuss the core technologies that were the basis for the contributions of this dissertation. Fourth, we elaborate on the key contributions that are presented in-depth in the next chapters of this thesis.

2.1 Motivating case studies

This dissertation was driven by multiple inspiring industry case studies that revolve around multi-organizational applications. By performing several case studies, we can elicit generic requirements that are used to drive a generic solution to the challenges that they pose. This solution, which is proposed as part of a contribution, is then validated in focused application scenarios stemming from these original case studies.

In order to substantiate the challenges that are addressed in this dissertation, this section introduces each case study and outlines the main issues that were encountered with regard to access control. Also, this section presents an overview

of some key issues that stem from the case studies and will be directly addressed by the contributions that are discussed in the next chapters.

The case studies cover a wide range of business domains, particularly electronic document processing, computer security monitoring services, workforce management, and electronic health record management. They involve a multi-organizational environment in which each organization is a different administrative domain that aspires pervasive control over authorization management. While these case studies expose a wide range of requirements with regard to authorization, we only focus on the requirements that are relevant to this dissertation.

2.1.1 Electronic document processing

The first case study involves a collaborative application that generates, manages, and distributes documents between organizations [75]. This application, referred to as *eDocs*, focuses on electronic work flows for business documents such as contracts, pay checks, invoices and reports. The application generates, manages and distributes these documents on behalf of enterprise customers such as financial institutions, insurance companies, and public utilities.

The eDocs application provides a shared platform to various organizations that may have conflicting interests. EDocs is operated on by a large set of *users*, which simultaneously perform actions on behalf of their employers. The employers require support for specifying authorization constraints that align with their organizational structure. Since it is also a document management platform, eDocs also enables sharing of documents between organizations directly, making it a *collaborative* application. This requires a system that supports management and enforcement of authorization configurations from the different organizations.

Consider an insurance company that uses eDocs for the management and distribution of invoices to its customers. These customers can be both private individuals and businesses that also use the document processing platform. In the latter case, these documents could be shared digitally, easing management and accelerating distribution. However, shared documents are subjected to confidentiality measures, as other insurance companies (which also may be using eDocs) should not have an insight in the clients of their direct competitors.

Hence, an evident authorization constraint would, generally, *restrict document reading to only users associated with the sharing organization, and of the receiving organization* **(R1)**[1]. This constraint may be specified by the application provider itself, and overridden by any organizations that need less restrictive

[1] A set of all requirements is provided in Section 2.1.5.

authorization constraints, albeit only for resources they own. This *isolation* constraint constitutes a key challenge that is addressed in this thesis.

Besides isolation, however, organizations require the ability to specify their own authorization constraints as well. For example, the insurance companies may want to restrict access based on a combination of roles, departments, responsibilities, and environmental properties (e.g., time of day). Moreover, these constraints should also be enforced when performing *data-driven* operations such as search queries. For instance, an office clerk of the insurance company may have the ability to search among the invoices that his/her employer sent out. However, a key challenge is to accommodate that *search results should also be restricted according to the privileges of the employee* **(R3)**. If the office clerk does not have the privilege to read an invoice, that resource should not show up in his/her search results.

2.1.2 Security monitoring services

Another case study involves an application that supports a security monitoring and compliance service. Security monitoring encompasses the collection, analysis, and response to monitored security events. The service aims at recognizing directed attacks and security breaches through continuous analysis of event logs from various data sources in the network of its enterprise customers. The identification of security incidents is performed by security analysts that are associated with the monitoring service, but the application also supports security experts associated with the enterprise customer to analyze the events.

Like eDocs, the security monitoring service requires *support for the enterprise customers to specify their own authorization constraints* according to their organizational preferences. We refer to this challenge as self-management **(R2)**. Enterprise customers must have full control over their data, requiring the ability to override and further restrict access of the security analysts to any resources that involve them. For example, these organizations may require a special clearance for security analysts to read event logs of firewalls that are operate in networks they are responsible for. The security analysts should also be restricted by authorization constraints specified by the monitoring service itself.

These *authorization constraints should also apply to data-driven operations such as aggregation* **(R4)**. For instance, security analysts should be able to see the total amount of potential security incidents per security category (e.g., network attacks, vulnerabilities), but only for enterprise customers that they have privileged access for. In other words, if a security analyst is not able to read incident reports of a certain organization, that incident report should not be aggregated in the analysis queries as to avoid the possibility of inference.

2.1.3 Workforce management

This case study is focused on a multi-organizational application in the domain of workforce management, referred to as *eWorkforce*. This section only elaborates on some core challenges. For a deeper analysis, the reader is referred to [76].

The application provides work flow planning for service appointments (e.g., install jobs) that is performed by enterprise customers (i.e., contractors). Such application entails multiple organizations with diverging goals. On the one hand, *workforce requesters* provide a set of tasks that need to be performed. For example, a cable company could submit a batch of install jobs for routers at clients in different locations. On the other hand, *workforce suppliers* are organizations that engage technicians that can be appointed these tasks. The eWorkforce application schedules an optimal planning of service appointments for each technician based on his/her availability, expertise, and location. It does this by taking into account various tasks from workforce requesters, and balancing them among multiple workforce suppliers.

Both workforce requesters and suppliers can benefit from outsourcing scheduling to a shared platform. However, security is a major issue, as the involved organizations may be competitors and the involved tasks expose the clients that need to be serviced. Consequently, a need for management and enforcement of authorization constraints is required. Due to the interactions and relationships of various organizations over the application, this is a daunting challenge.

Consider an authorization constraint in which *technicians can read the service appointments of the workforce supplier they are associated with, if the industry branch of the workforce requester associated with the appointment aligns with the expertise of the technician* **(R5)**. This constraint considers not only the relationship that the technician has with the workforce requester, but also evaluates properties associated with both technician and workforce requester. There is a need to accommodate specification of such constraints in a concise and straightforward manner.

2.1.4 Electronic health record management

A last case study encompasses the management and sharing of medical records among health institutions. Enabling applications must evidently apply a large set of security technologies to achieve this task, but our focus is on authorization management.

Electronic health applications generally involve many parties that should have different privileges to various sets of data. In particular, end-users may fulfill

any role ranging from patients, nurses, secretaries, GPs, specialists, or trainees, which may influence their privileges.

The diversity of parties must be accommodated for in the authorization constraints. In order to specify such constraints on medical records, (in)direct relationships between these parties should also be taken into account. For example, consider a medical professional that attempts to read a medical record of a patient (e.g., the report of a consultation). An authorization constraint could be that *medical professionals can read medical records if the patient of its corresponding consultation already had a consultation with the acting professional's (in)direct supervisor in the past* **(R6)**. In this authorization constraint, supervisor is a *recursive* relationship over which properties are evaluated. For example, the preceding constraint involves assessing the consultations of the supervisors along the path, and confirming whether there exists one that corresponds with the targeted patient.

2.1.5 Challenges

Table 2.1 presents a summary of the challenges that were suggested by these case studies. These challenges outline the functional requirements we wish to support in the contributions of this dissertation. Challenges R3 and R4 correspond to a need for comprehensive enforcement, and challenges R5 and R6 correspond to the need for organization-centric expressiveness. While they were illustrated by an example of a specific case study, they are more generally applicable and different examples laying out these challenges can be found in all of the discussed case studies.

Specifically, the relationship-bound challenges (R5) and (R6) expose an issue that can be regarded from two perspectives. First, enabling property comparisons over relationships aligns well with the objectives of multi-organizational applications, as they enable reasoning in terms of how the organizations must be related and to what properties they must appease in the authorization constraints. Second, this supports more expressive authorization constraints, which is a necessary prerequisite in enabling run-time customization in a systematic way. This is done through *policy-based access control*, which is discussed later in this chapter.

These challenges concretize the functional requirements that we address in this dissertation. However, the case studies also exhibit a set of non-functional requirements. These were already outlined in Chapter 1, and are common for all the analyzed cases. Particularly, performance and reusability are important non-functional requirements that will yield special attention in the architecture of our contributions.

Table 2.1 – The case studies introduce challenges which were summarized in Chapter 1. Challenges R3 and R4 encompass the need for comprehensive enforcement while challenges R5 and R6 contribute to the organization-centric expressiveness requirements.

	Challenge	**Description**
R1	Isolation and sharing	Restrict resource access according to organizational association, but support exceptions if preferred by the provider or owning organization, even on an individual basis.
R2	Self-management and multi-layered management	Empower organizations to specify their own authorization constraints according to their organizational needs. Also, all involved organizations, including the service providers, should be supported to perform fine-grained authorization management.
R3	Search restriction	Restrict results of search queries to elements to which users are privileged.
R4	Aggregation restriction	Include only elements in a data aggregation operation to which the requesting user has privileges.
R5	Relationship properties	Enable security administrators to specify authorization constraints on the existence of relationships, and comparable properties associated with entities in these relationships.
R6	Relationship quantification	Support authorization constraint specification over *recursive* relationships, in which properties associated with entities along the relationship path can be compared.

To discuss these contributions in more detail, this chapter first elaborates on the background concepts that underpin them in the next section. This is followed by an outline of the core technologies of this work, in order to finally describe the contributions themselves in the last section.

2.2 Background in Access Control

This section provides a general overview of access control and discusses research domains related to this thesis. First, it introduces core concepts of access control. Next, it elaborates on prominent access control models. This is followed by a discussion of policy-based access control. Finally, it outlines the research domain of authorization in multi-organizational applications and discusses some related domains in collaborative applications, grid computing, cloud computing, and federated authorization.

2.2.1 Access Control

This section discusses the core concepts of access control, and provides an overview of the fundamentals of management and enforcement of authorization constraints.

In general, access control consists of three components: authentication, authorization, and audit [203]. Their interaction is illustrated in Figure 2.1. Authentication verifies the identity of the user, authorization restricts the action that the user attempts to perform, and audit collects and analyses user activity.

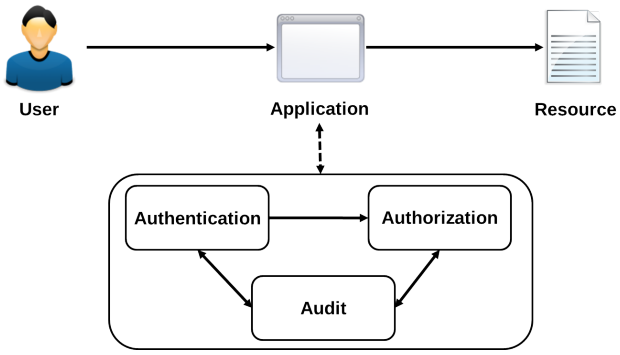


Figure 2.1 – Access control consists of three stages that are performed on each user request to the application.

All components should be performed on each request that a user performs on an application, either implicitly or explicitly. Also, depending on the system, an interaction between the three components may exist.

Authentication and Audit

Authentication regards the verification of the identity of the user that interacts with the system. Although the most visible form of authentication is a user login through user name and password, various other authentication forms exist. These are generally summarized into three categories: what you know (e.g., a password), what you have (e.g., certificate or session key [98]), or what you are (e.g., biometrics [18]). These categories can be combined to harden the authentication process, effectively establishing multi-factor authentication [105]. Another method to harden authentication, also depicted in Figure 2.1, is *continuous authentication* [96], which constitutes extensive interaction between the authentication and audit component to validate the user identity, typically based on behavioral characteristics [183]. While authentication has traditionally been performed on a per-application basis, single sign-on systems are gaining attention both in research [154, 222] and in industry [123, 198].

Audit regards the collection, organization, and analysis of authentication and authorization events [126]. This can be used to detect anomalies both during and after the authenticated session. Audit can also be closely knit into the authorization process, providing a system of a posteriori compliance control of the actions that a user previously performed on certain computer resources, based on a justification provided by that user during authorization [53]. Similarly, break-the-glass [44, 92] ensures non-repudiation of actions that override authorization constraints, in order to accommodate reliable access to essential data in emergency situations. Audit analysis is most commonly known on a network level, in particular for intrusion detection systems [14, 163]. However, its importance in a general sense cannot be understated [203].

Authorization

Authorization regards the restriction of the actions that users can perform on computer resources in a certain context [205]. Such restrictions are typically applied on every request that a user makes to a system through implicitly or explicitly permitting or denying access by enforcing a set of authorization rules. Authorization is the main focus in this thesis. In particular, we focus on a subset called *application-level authorization*. Since the term *access control* is also commonly used to refer to authorization in literature, we use these terms

interchangeably throughout the text. Unless explicitly stated otherwise, they both refer to the term *authorization*.

Concept. In essence, authorization can be seen as the application of a set of rules that regards a subject, object, action and a context in order to determine whether to permit or deny a certain access request. In this regard, *subjects* are typically application users. However, system processes and services may also serve as subjects that perform an access attempt *on behalf of* a user[2]. On the other hand, *objects* (or *resources*) are the entities that an access attempt is directed at. These can be any form of software resource, ranging from documents, database entries, to even subjects themselves.

Authorization concerns three aspects: *confidentiality*, *integrity*, and *availability*. Confidentiality prevents unauthorized disclosure of information. Integrity impedes improper malicious modifications to the data. Availability assures access for authorized entities to the information. While ideally, an authorization system should ensure all three of these aspects, this often proves non-trivial to achieve without making the authorization system too rigid.

Enforcement. The enforcement and decision process for an access control decision is realized by a concrete *security mechanism*, typically through a *reference monitor* as illustrated in Figure 2.2.

Figure 2.2 – A reference monitor evaluates authorization constraints for access attempts, determining whether a subject is permitted to perform a certain action on a resource in a certain context, and enforcing this decision.

The *reference monitor* is a trusted computing base component that mediates every access of a subject to a resource according to a set of predetermined authorization constraints [25, 141]. Such constraints can be statically defined (e.g., permissions) or dynamically acquired (e.g., delegation or through subject properties). This component should be tamper-proof, non-bypassible and suspectible to verification methods [200].

Management. Authorization constraints first need to be specified prior to enforcement. Privileges stemming from those constraints can be established in various ways, and we refer to these approaches as the *management classes*.

[2]In this thesis, we will use terms *subject* and *user* interchangeably, although in some situations, especially when focusing on information flow, a distinction can be made.

These classes determine *who* composes the authorization constraints. Generally, access control is divided into two management classes: discretionary access control and mandatory access control.

Mandatory Access Control (MAC), is the management approach that regulates authorization constraint specification through a central authority. MAC-authored policies are typically *property-driven*, i.e., they contain a set of authorization constraints that relate properties associated with the resource and user in order to make an authorization decision. For example, such properties could involve security clearances held by the user. MAC is usually associated with the concept of *multi-level security* [24], which uses a lattice-based access control model to issue categories and clearances to resources and users, respectively, in order to centrally control the authorization of resources. Driven from a primarily military context, this management approach enforces strict control on the flow of information, but is typically considered too rigid for commercial application domains.

Discretionary Access Control (DAC) alleviates this issue through an *identity-driven* approach that is generally coupled with an administrative policy that enables propagation of access constraints. Users can be given the ability of granting or revoking privileges to other users through administrative policies. DAC provides a more flexible and intuitive approach to specify authorization constraints. However, this approach does introduce some issues, particularly with regard to information leakage and management capabilities. The latter is caused by the identity-driven focus of the approach, which inherently does not scale with large sets of resources and/or users.

In order to alleviate issues with both approaches, their characteristics are often combined through conducting a discretionary approach within the boundaries of MAC. Several access control models, particularly role-based access control [90], assume a combination of both approaches as a means of reducing the drawbacks. In multi-organizational applications, such a composite approach also seems reasonable, as it involves a central authority (i.e., the application provider) that constrains capabilities that may be further restricted according different administrative domains (i.e., individual enterprise customers). The enterprise customers want to enforce their own authorization preferences, and administrative policies should enable their security administrators to do so. However, this should be done within clear bounds determined by the application provider.

2.2.2 Access Control Models

Access control models provide a means to specify authorization constraints in a unified and formalized manner. As opposed to management classes that determine *who* establishes the privileges, access control models provide a framework that facilitates security administrators to reason about *how* permissions for actions between subjects and resources are organized.

Over the last decades, several access control models have been proposed. This section provides an introduction into five prominent models: identity-based access control, lattice-based access control, role-based access control, attribute-based access control, and relationship-based access control.

Identity-Based Access Control

Identity-Based Access Control (IBAC, also referred to as *permission-based access control*) has been a cornerstone in pure DAC management approaches, and focuses on the explicit assignment of authorization permissions to users.

IBAC can be conceptualized through an access matrix [113], in which users are represented as rows, resources as columns, and privileges (i.e., permitted actions) as the contents of the cells. Figure 2.3 depicts an example access matrix. If a user was not assigned a permission to perform an action on a specific resource, the access attempt for that action on that resource is denied.

Access matrices are typically implemented through access control lists or capability lists. An access control list is stored for each resource, and specifies the users and actions that comprise the permissions for that resource. Similarly, capability lists are stored for each user, and specifies which actions that user can perform on which resources.

While IBAC provides an intuitive approach in specifying authorization constraints, it does not scale well with regard to the number of users and resources in the system. As a result, identity-based specification quickly becomes unmanageable when authorization controlled through a (semi-)centralized

	$Resource_1$	$Resource_2$	$Resource_3$
Subject A	`read`	`read,write`	`execute`
Subject B	`execute`	-	`read`
Subject C	`read,write`	-	`execute`

Figure 2.3 – Access matrices provide a framework to represent the privileges for each user on the resources in a system.

authority. While presence of administrative policies may alleviate this issue to a certain extent, this model is generally regarded as cumbersome for centralized authorization management.

Lattice-Based Access Control

Similar to IBAC, Lattice-Based Access Control (LBAC, [204]) has been identified with pure mandatory management approaches since its origins. In the most general sense, LBAC associates a single property, such as a security class, with its resources. On these security classes, a partial order exists that is modeled to regulate information flow in an organized manner.

The most well-known example of a lattice-based policy is the military lattice that associates `top secret`, `secret`, `confidential` and `unclassified` security classes with its resources. Similarly, users are associated with a certain security level corresponding to these classes, and information flow is controlled through having the subject associate a user session with the highest security level they need to achieve their goals. Through this scheme, confidentiality can be mandated. The policy model for this approach was formalized by Bell-Lapadula [24]. Biba proposed a dual approach to ensure integrity [32].

The lattice-based model is a rigid system that is well-fit for centrally controlled authorization administration that is usually associated with MAC. While it can be used to ensure a certain confidentiality and integrity level of the data, this model is generally seen as too inflexible for many application domains [61].

Role-Based Access Control

Role-Based Access Control (RBAC) is an identity-driven access control model that uses roles as an indirection between subjects on the one hand, and *permissions* (i.e., actions and a specific (group of) resources) on the other [90]. In RBAC, subjects are assigned to a (set of) roles, and roles are assigned to a set of permissions. This is depicted in Figure 2.4.

With regard to management, RBAC combines both mandatory and discretionary principles [180] through associating an authentication session with a predetermined subset of the roles that were assigned to the subject. RBAC can be both centrally administrered and partially *delegated* through administrative policies [202]. A lot of work was previously performed to study role delegation (among others, [69, 114, 130]) and the use of administrative policies to perform role management (among others, [80, 239]).

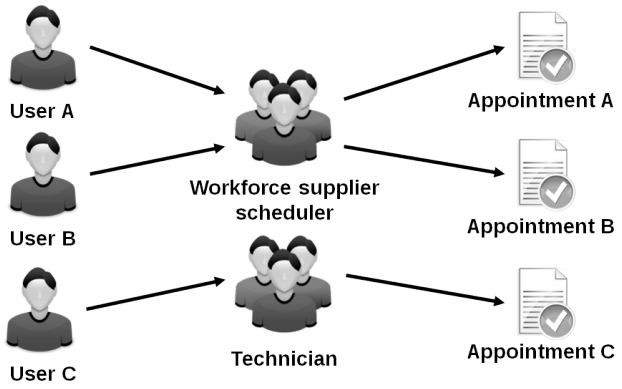


Figure 2.4 – Role-Based Access Control. Roles serve as an indirection between subjects and permissions, improving management scalability. Example for the workforce management case.

RBAC introduces two interesting concepts: role hierarchies and separation of duty. Role hierarchies define a partial order on roles that enable permissions assigned to parent roles to be inherited by their child roles. This concept simplifies management, as it supports a more modular aggregation of permissions through inheritance. Separation of Duty (SoD) partitions tasks and associated permissions among different roles to prevent a single subject to acquire too much privileges. It was inspired by commercial requirements such as the Chinese wall policy [42], but the concept allows for a much broader range of restricting access for subjects.

RBAC identifies static and dynamic SoD classes to constrain privileges. Static SoD enforces restrictions on the assignments of roles to subjects. For example, if a subject is assigned a role A, it cannot be assigned a role B. On the other hand, dynamic SoD provides operational flexibility by restricting the roles that a subject can be simultaneously associated with during an authentication session. For example, a user may be assigned both roles A and B, but is not permitted to exercise the permissions associated with both roles simultaneously.

Over the past two decades, a lot of research has been conducted that elaborates on role-based access control [99]. These comprise works on role engineering [150] and mining [138] methodologies, to applied technologies and middleware architectures [47, 170] and case studies [206].

While RBAC has become a popular authorization model, it does have several issues. In particular, it is generally not well supported to express ownership-based constraints (e.g., that the creator of a file can access it) and also suffers

from *role explosion* [85]. Role explosion encompasses the creation of a large amount of roles, typically because of specialization of responsibilities, to an extent that it becomes an administrative burden [206]. Also, since RBAC is identity-driven, it draws heavily on the explicit assignment of roles to users. In *open systems*, such as web applications, the users are not known a priori, and role assignment can again cause administrative overhead. Moreover, the inflexibility of the standard model to deal with slight variations in authorization constraint specification has driven the proposal of a large set of extensions over the years [2, 31, 72].

Attribute-Based Access Control

To cope with these issues, Attribute-Based Access Control (ABAC, [119]) has drawn increasing attention. ABAC is a property-driven model in which attributes are assigned to subjects, resources, actions, and the environment (e.g., to represent the current time). In contrast to the previous models, authorization in ABAC is determined dynamically [207] based on the attribute values corresponding to the related subject, resource, action, and environment of an access request. For example, a document in the eDocs application from Section 2.1 may be assigned a specific identifier associated with the subject that owns it. Moreover, resources may have certain categories as their attributes, and privileges could be determined based on the competence of the subjects to perform operations on documents of that category.

While several formalization attempts have been proposed [33, 131, 208], no consensus currently exists as to what precisely constitutes ABAC, although aforementioned concepts are typically regarded as part of the model. In research, attention in ABAC has been focused on several aspects [209]. In particular, attribute mining and engineering poses a considerable challenge that has kindled several works [160, 238]. Additionally, widespread attention has been drawn onto the migration from existing models on how to integrate them into ABAC, particularly the use of roles [132, 137].

While ABAC is an emerging mechanism for specifying authorization constraints, it does have some major issues. First, attributes that are taken into account for an authorization decision may have questionable provenance [201], therefore authorization systems need strong trust relationships with the authorities providing the attributes. Second, in order to focus research attention, consensus on the core concepts that constitute ABAC is needed. Third, the current consensus of ABAC does not adequately support relationships of subjects and resources to influence privileges. For example, it is not straightforward to specify a rule that restricts according to the *industry of the owner of a document* that an

access attempt is made on, as this involves a separate attribute that reflects the industry of the resource owner, which only implicitly reflects that relationship.

Relationship-Based Access Control

Relationship-Based Access Control (ReBAC) expresses authorization constraints in terms of relationships between subjects and resources [93]. It was driven by the popularity of social networks [48, 52, 59], but has since been argued to be applicable to other application domains as well (e.g, in eHealth [5]). Since its inception, several models have been proposed (among others, [46, 70, 117]).

In essence, ReBAC identifies a relationship between subjects and resources, and specifies authorization constraints based on them. For example, a subject may have a relationship *"creator"* with a certain document, and the authorization constraints may explicitly permit any requests if a relationship of this type exists between the subject and the resource. On the other hand, this constraint by itself does not imply the subject to have access to resources with which other types of relationships exist.

ReBAC is an emerging authorization methodology that is gaining attention in research. However, one major issue is that not every aspect can be naturally expressed as a relationship. This includes, but is not limited to, temporal constraints (e.g., access is only permitted during working hours). Moreover, it is currently unclear to what extent this model is generally applicable to common authorization specification requirements and real-world cases.

2.2.3 Policy-Based Access Control

With many organizations involved, authorization constraints evolve at a faster pace than the functionality of a multi-organizational application. A mechanism is needed that supports modifying these constraints without a need for recompiling the application code. One approach to address this is through Policy-Based Access Control (PBAC, [200]).

In essence, PBAC enables modifying authorization constraints independently from the application code through externalization of the authorization specifications into a separated artifact called a *policy*.

PBAC aspires a maximal modularization of access control, in which a subject request to the application is mapped onto a security policy at run-time in order to evaluate whether the intended action is permitted.

This provides several advantages. First, it supports changing the authorization model independently from the application. Second, it supports separation of concerns [74]. Application developers need not involve themselves with the specification of access control policies, as the responsibility can be delegated to security administrators. Third, it supports run-time reconfiguration of access control policies. This enables customization independently from the application code. Consequently, security administrators can modify the policy whenever business needs change without requiring application redeployment. Fourth, it facilitates multiple parties to formulate their own security requirements on a shared platform, thereby enabling multi-organizational applications to be secured in a customizable manner. Fifth, it promotes analysis, verification, audit, and testing of authorization constraints because of the centralized approach of policy management.

This section introduces general characteristics for PBAC and outlines some of the research contributions that it has received. It first elaborates on the concept and introduces a reference architecture for PBAC systems. Next, it describes how policy languages achieve specification of authorization constraints and discusses how this kindles research efforts in the realm of analysis, verification, testing, and evaluation performance optimization. Finally, this section provides a short elaboration of how enforcement can be instrumented as part of the application.

Concept and framework

PBAC seeks the externalization of authorization constraints from the application code. As such, a *policy* specifies these constraints and needs to be evaluated and have the authorization decision enforced. The process of authorization in PBAC has been analyzed extensively, and a reference architecture has been proposed [232] to streamline this process and lead standardization efforts. Figure 2.5 demonstrates the authorization process for PBAC systems and outlines communication between different components in the process.

The reference architecture consists of six components. The Policy Enforcement Point (PEP) is the component that composes and forwards authorization requests and enforces the authorization decisions. The Obligation Service (ObS) performs any obligations mandated by the authorization policy. Obligations are actions that must be executed alongside enforcement (e.g., log an event). The Context Handler (CH) coordinates messages that are sent internally and externally to the authorization system. The Policy Information Point (PIP) handles requests for properties and retrieves them from the relevant data systems. The Policy Decision Point (PDP) performs a policy evaluation, and

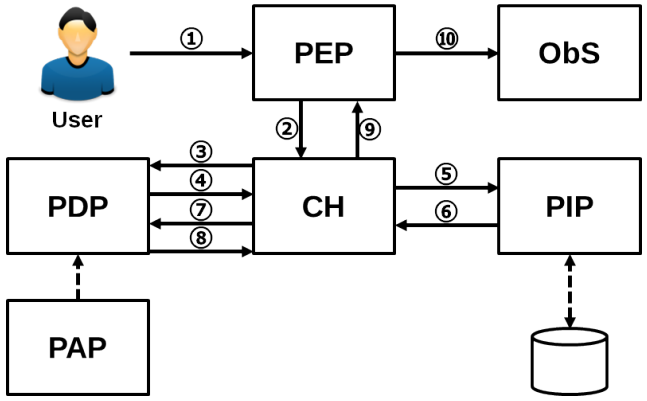


Figure 2.5 – The authorization process for the reference architecture in policy-based access control.

composes an authorization decision that includes relevant obligations. The Policy Administration Point (PAP) handles management and retrieval of authorization policies.

As shown in Figure 2.5, the authorization process is preceded by the PAP provisioning the relevant policy to the PDP. Any request by a user is intercepted by the PEP (1), which composes an authorization request and forwards it to the CH (2). The CH then indicates to the PDP to initiate evaluation (3). The PDP may request additional attribute values to be retrieved to the CH (4) which is forwarded to the PIP (5). The PIP returns the requested attribute values (6) via the CH to the PDP (7). The PDP uses these values to compose an authorization decision and send it to the CH (8). The CH composes an authorization response to the PEP (9) which forwards obligations to the ObS and enforces the decision (10).

The reference architecture proposes a modularized and access control model-independent approach to evaluate and enforce policies. This introduces challenges in how to specify the authorization constraints, enforce the decisions, and support analysis of policies by security experts.

Languages

Authorization constraints in policies are expressed through the use of policy languages. Policy languages, to some extent, can determine the access control model that is imposed on the policy and are recognized by several special characteristics. A wide range of policy languages have been proposed,

both in research (among others, SPL [193], Ponder [73], ASL [127]) and in industry (among others, SecPAL [22], EPAL [12], XACML [172]).

The design of a policy language depends on a wide range of characteristics. These characteristics are often influenced by the purpose of the language (e.g., privacy [23] or general authorization [172]). Some of the characteristics include what constitutes the focal point for expressions (e.g., roles [73], attributes [172] or entities [193]), rule semantics and conflict resolution, and size of the decision set. While a wide range of policy languages have been proposed with different characteristics, this dissertation is primarily influenced by XACML.

XACML. The eXtensible Access Control Markup Language (XACML, [172]) is an OASIS standard directed towards specification of authorization policies in an XML format.

The language is recognized by three core concepts: attribute-based expressions, multi-valued decision sets, and policy trees.

- XACML considers attributes as first-class citizens in its expressions. In these *attribute-based expressions*, the attributes are compared with each other and with concrete values. Expressions are evaluated to determine the applicability of rules and policies.
- Rules are the cornerstone of the policy evaluation. An applicable rule can influence the final evaluation result through its effect. A rule evaluates to `Permit`, `Deny`, `NotApplicable` (if the expression corresponding to that rule is not satisfied for a certain evaluation context), or `Indeterminate` (in case of evaluation errors). The latter is further subdivided in multiple decisions that instruct further handling to the decision engine. An evaluation context indicates at least the user, resource, and action that are related to a particular authorization request.
- While a rule can influence an evaluation decision through its effect, multiple rules can be applicable in the same policy. However, because XACML supports both permit and deny effects, this can lead to conflicts that must be resolved. This *conflict resolution* is done through a combining algorithm, which prioritizes based on decisions of the rules or order. For example, a *permit overrides* combining algorithm prioritizes any permit decision that is encountered during evaluation of the rules over other evaluation results.
- Rules are organized as part of policies. Policies contain a target for determining its applicability at evaluation time, and a combining algorithm to resolve possible conflicts of its children. Similarly, policy sets can contain other policies or policy sets as children, thus forming *policy trees* that support modular specification of constraints.

Because policies must be specified in an XML-based format, policy specification can become cumbersome. This has driven design of authorization languages that have the same policy model, but support more straightforward specification [79, 102, 169]. Notably, STAPL [79] is a language that is based on the XACML policy model. It is considered a core technology of this dissertation and is further discussed in Section 2.3.

XACML is considered the de-facto standard for attribute-based policies. However, its expressiveness also introduces considerable complexity in terms of policy specification. This has led to a wide range of research efforts in both analysis [129, 134, 228] and testing [27, 156, 236] to determine whether the policy corresponds to the original intentions of the security administrator.

Moreover, the overhead incurred by the policy evaluation process [227] has also driven research towards performance optimization of PDPs. Approaches such as decision caching, policy refactoring, decision engine optimization, and efficient attribute fetching have been proposed to mitigate this issue. In particular, decision engine optimization is done through a wide range of techniques including numericalization [146], decision diagrams [168, 185], and set algebra [165].

Enforcement

While policies can be used to indicate authorization constraints, they still need to be enforced in the application in order to effectively restrict access. As discussed earlier in this section, authorization enforcement is performed through PEPs. These can be implemented in various ways depending on the authorization framework that is used to do the specification.

Enforcement *hooks* are implicit or explicit calls to the authorization subsystem. They are typically provided through conditional structures in code, although code injection techniques such as AOP have also been used to achieve hook placement [231].

A common problem occurs when these hooks are not placed systematically, leaving part of the application functionality exposed to unauthorized access [230]. Research efforts have sought to alleviate this issue through identification of placement candidates and automated hook placement frameworks. Both static and dynamic code analysis techniques have been employed to identify sensitive operations and resources in the application. These techniques are based on recognition of security-sensitive library calls [166, 216] (e.g., a read to the file system), through attempting access to functionality discovered with privileged users with unprivileged roles [220], or fingerprinting through concept analysis [100]. However, an approach unified across languages and domains that

effectively detects hook absence in security-sensitive functionality is still lacking. Since these approaches are orthogonal to the contributions in this dissertation, they are not elaborated on further.

2.2.4 Access control in multi-organizational applications

This section regards the research that has been performed for access control in multi-organizational applications. When regarding multiple organizations that have different administrative domains, access control management can become a complex task. This section regards several research domains related to access control in multi-organizational applications. In particular, it elaborates on access control approaches in collaborative applications, grid computing, cloud computing, and federated access control.

Collaborative applications

The domain of access control in collaborative applications has been extensively studied in the past. Collaborative environments recognize the need for sharing resources among various cooperating organizations. These organizations comprise of separate administrative domains that want to enforce their own policies. As a consequence, the emphasis in the research domain has traditionally focused primarily on the authorization management in collaboration coalitions.

To this extent, the focus has historically been on extending role-based access control models to accommodate this issue [43, 56, 118, 151, 226]. This has been done through (i) introduction of a centralized administrative authority policing the sharing of the involved administrative domains [143, 243], (ii) decentralized authorities that perform delegation to achieve a collaborative environment [130, 242], or (iii) mapping of roles between administrative domains [19, 210].

As for the first approach, several works have appeared that support management through a central authority. In particular, Robac [243] assigns explicit ownership of resources to organizations, and privileged access implies being member of that organization and being assigned the proper roles. Another method was proposed by Li et al. [143], which proposes a *group-based* RBAC model that supports authorization management both on a system-level and on a group-level. Users are assigned to collaboration groups, these groups support specific roles that are managed by designated group administrators, decentralizing administrative tasks to some extent.

In the context of the second approach, Ramars [130] supports ad-hoc collaborative sharing through decentralized management authorities through

privilege delegation. Roles are independently defined across multiple administrative domains in a distributed environment, and privileges can be delegated to decrease the administrative burden. This work also models some special role types and capabilities to accommodate dynamic collaborative environments.

The third approach focuses on mapping roles between different administrative domains in order to support multi-domain security interoperability [19, 210, 211]. Among others, Baracaldo et al. [19] follow this approach through a system that supports security administrators to perform queries on permissions, and determine the appropriate roles to map on according to the permission sets. As recognized in this work, a general challenge to this approach is coping with the complexities of separation of duty constraints, and a great deal of attention is focused on ensuring proper enforcement.

Several works have also appeared that apply attribute-based policies to collaborative systems [215, 245]. In particular, MPABAC [144] provides a model and architecture that regulates XACML policies originating from various administrative domains that are locally evaluated. Lin et al. [145] deal with the management issues through decomposition of a cross-organizational XACML policy. The policy is decomposed and evaluated at trusted sites. Next, the individual decisions are again combined at the original decomposition point to come to a definite decision. Mazzoleni et al. [157] propose policy integration algorithms as an extension to XACML to deal with integrating policies stemming from different administrative domains.

Relationship-based access control policies have also been applied to the domain by various works [49, 71, 121]. These works have put a strong emphasis on access based on relationships in on-line social networks, but their contributions are more generally applicable. Notably, COLLAC [71] makes use of the hybrid logic model proposed by Bruns et al. [46] in order to map users to resources. This mapping is used as a cornerstone to specify restrictions in the policy. Paci et al. [181] provide an overview of access control in *community-centered* collaborative systems. This work is primarily focused on, but not limited to, the application of ReBAC in collaborative systems.

While the main focus in collaborative application authorization has been directed towards management, attention has also been drawn to the systems that enforce it. Notably, xDAuth [7] is a framework for cross-domain authorization and delegation. A trusted delegation service determines the privileges for any cross-domain access requests, and serves as a central policy decision point to which policies from different administrative domains are published.

Grid computing

Grid computing encompasses the sharing of storage and computing resources between different virtual organizations through open protocols and interfaces. Virtual organizations can be either physically distributed institutions or logically related groups [95]. Each virtual organization represents a separate administrative domain that wants to determine their own authorization preferences.

In order to control access to the resources in a grid system, one challenge is dealing with heterogeneity of the authorization management and enforcement systems of involved virtual organizations. This can be addressed either by centralizing authorization management, or decentralizing it [213].

The decentralized approach was initially focused on applying authorization on the level of virtual organizations as a whole [94]. Users either had access to all nodes of a virtual organization, or to none at all. A need for more open and dynamic authorization infrastructures led to specification and enforcement on a node-level (e.g., PERMIS [55] and PRIMA [149]). Particularly, PERMIS proposed a role-based authorization system that supports history-based decision making and separation of duties. The system put a strong emphasis on distributed credential management to achieve this. They also supported coordination between authorization decisions made by different parties in a more centralized fashion, primarily in support of separation of duty policies across the grid domain. One problem with (predominantly) decentralized approaches is that they consider policy enforcement at grid nodes. Consequently, brokering services that allocate resources are unable to efficiently take into account security constraints [158]. Moreover, these approaches suffer from the complexity of mapping of attributes between different administrative domains.

A more centralized approach alleviates this issue at cost of full independence of the administrative domain and availability. For example, VOMS [9] proposed a single administrative domain through which privileges were managed through role-based access control for each subject. The system also elaborated on delegation of administrative rights to alleviate the inflexibility and administrative burden to some extent. Mazzoleni et al. [158] take a different approach that supports both global (i.e., grid-wide) policies, and local (i.e., node) policies. that apply to their shared computer resources. Nodes publish these local policies to their administrative domain controllers to support conflict resolutions with regard to the global policy. The resource broker that is associated with the administrative domain handles allocation of resources for the specific nodes in their domain, taking into account local authorization restrictions. While a centralized approach eases the complexity of allocation and mapping to some

extent, centralized systems do require a trust relationship to exist between the involved virtual organizations and the central authority.

Cloud computing

Cloud computing supports the outsourcing of IT resources by organizations (i.e., *tenants*) to a shared platform. Some of the key characteristics of cloud computing are the rapid elasticity of computing capabilities and resource pooling to serve multiple consumers in a multi-tenant model [161]. The latter property leads to a need for an approach that properly separates each tenant through access control. As such, one major focus for research in cloud computing has been on authorization management in order to ensure separated data access for multi-tenancy.

Multi-tenancy deals with supporting separation of management and privileges among tenants (also referred to as *tenant isolation*). In cloud computing, management is typically regulated by the service provider, which serves as a central trusted authority. The service provider accommodates the authorization preferences for the involved tenants and ensures they are properly enforced.

Various works have focused on the authorization management in cloud computing, including role-based (e.g., [8, 246]) and attribute-based (e.g., [186, 192]) adaptations to existing access control models that take into account multi-tenancy. In [223], the MTAS system does this through explicitly modeling issuers (which represent the tenants). The model also supports granting of permissions to roles associated with another issuer in order to support sharing between tenants. In [186], this concept is extended to attributes, which are uniquely owned by tenants. Tenants are responsible for assigning the values for the attributes they own, also supporting inter-tenant sharing of resources. AC3 [240] is a multi-layered model that takes into account roles, tasks, security classifications and explicit permissions to protect access and information flow. However, they do not elaborate on enforcing multi-tenancy requirements such as tenant isolation.

Chapter 3 will elaborate further on several works in this research domain, as it is focused towards authorization management and enforcement in multi-tenant Software as a Service (SaaS) applications.

Federated Access Control

Federated authorization focuses on decentralizing both management and enforcement of authorization policies in the administrative domains for federated

applications [133]. Similar to federated authentication technologies such as OpenId [198] and SAML [171], it focuses on security enforcement by a trusted authority, typically in the administrative domain of the participating organization.

The research domain stems from that of Service Oriented Architectures (SOA), which also focused on decentralization of administrative domains, open systems (i.e., the participating subjects are not known a priori), and with a strong emphasis on trust relationships. Because of the focus on open systems, SOA research has been the main driver for attribute-based access control [54, 241].

Menzel et al. [162] proposed an authorization architecture in which the local authorization system of the service requester domain could determine whether the subject's request was permitted. This required a strong trust relationship between the requester domain and the service provider, as the assertion of this privilege was sufficient for granting access. Such a trust relationship was also required in [11]. This work also proposes a federated authorization system in which anonymous credentials are used for claims of conforming to an authorization policy. Decat et al. [78] describe an architecture for federated authorization in multi-tenant SaaS applications which supports policy evaluation both at the service provider and at the tenants.

Industry standards have also contributed to federated authorization initiatives. In particular, Kerberos [13] and SPKI [86] support claims on privileges in a distributed environment. OAuth [112] is a web standard that supports client organizations to access a subset of the resources through authorization tokens granted by its owner. While these initiatives reduced the burden of authorization at the service provider and enable authorization outsourcing, they generally require a strong trust relationship to be in place between the participating parties. Moreover, federated authorization may impact performance prohibitively for certain applications.

2.3 Core technologies

The contributions in this dissertation are supported by several core technologies:

- **Attribute-based access control.** Attributes that are assigned to subjects, resources, actions, and the environment enable fine-grained rule specification. These fine-grained rules can be incorporated in various ways to address the challenges introduced in the beginning of this chapter, and can improve authorization management.

- **Policy-based access control.** Externalization of authorization constraints from the application code into a separate policy enables run-time reconfiguration of authorization preferences, facilitates specification of such preferences by involved organizations, and generally increases flexibility of the authorization system.
- **XACML policy model.** The XACML policy model introduces policy trees, multi-valued logic, and attribute-centric expressions as part of a policy language. These concepts are leveraged to address the challenges of the beginning of the chapter.

These technologies are accumulated as part of the STAPL policy language [79]. While STAPL has a policy model that resembles that of XACML, it simplifies policy specification as it is not based on an XML format. Moreover, it simplifies some of the structures of the XACML model, which reduces complexity without loss for generality. The remainder of this section elaborates on the model of STAPL. It also provides an example STAPL policy.

STAPL

STAPL is a declarative policy language that supports authorization specification through subjects, resources, actions, and the environment. Attributes assigned to these elements can be compared to each other and with concrete values in expressions, which are composed into rules and policy targets. It was incepted as a domain-specific language in the Scala programming language. While the policy language was influenced to a great extent by the XACML policy model, it does have several minor differences such as the specification format, policy structures, and supported combining algorithms. The core concepts of STAPL can be explained through the policy structure, expressions, and evaluation.

Policy structure. STAPL supports two types of *policy elements*: policy components and rules.

A *policy component* resembles the policies and policy sets of XACML. It consists of (i) a target expression, (ii) a non-empty, ordered list of policy elements, and (iii) a combining algorithm to resolve decision conflicts between its direct children. Because a policy component can have other policy components as its children, *policy trees* can be formed that organize security concerns across the policy in a modular fashion. Supported combining algorithms include `permit overrides` (which prioritizes permit decisions of applicable child elements during evaluation), `deny overrides`, and `first applicable` (which prioritizes decision according to the order of the child policy elements).

On the other hand, *rules* consist of a condition expression and an effect. The effect (`permit` or `deny`) indicates which decision must be taken into account for the rule if the condition is satisfied. Rules comprise the leaf nodes in the policy trees, and ultimately provide the decisions that need to be enforced.

Expressions. Central to STAPL policies are the boolean expressions that are part of the policy components (through *targets*) and rules (through *conditions*). These boolean expressions compare attributes assigned to subjects, resources, action or environment to each other or to concrete values. Alternatively, they compound other expressions through conjunction, disjunction, and negation. The latter enables composition of complex expressions that take into account various comparisons.

Policy evaluation. Whenever a policy is evaluated, it is done so for a concrete subject, resource, action, and context (i.e., the environment) that embodies the access request for which a decision must be made. We refer to this as the *evaluation context*.

Evaluation is performed through substituting attribute references in the expressions with concrete values according to the relevant principals of the access request, possibly by fetching these attribute values from an underlying data source. If the target expression is satisfied for a certain policy component, all children of that component is evaluated.

For policy components, this entails evaluating their targets and recursively repeating this process until a leaf is evaluated (i.e., a set of rules are reached), or until no direct children are found to be applicable. For rules, this entails evaluating the *condition*. If the condition of a rule is satisfied, its corresponding effect is taken into account and possibly combined with other decisions of its direct parent policy component. Through recursive evaluation, decisions of the applicable elements in the policy trees are combined to ultimately come to a single decision.

This is illustrated in Figure 2.6. In this figure, rule r_6 ultimately contributes the decision in this evaluation, as it is decisive for p_3 and overrides the decision of r_3 due to the `permit overrides` combining algorithm in p_1. Rules r_2, r_4, r_5 and r_7 are not applicable in the evaluation context.

A policy element thus evaluates to one of three decisions: `permit`, `deny`, or `not applicable`. A rule is found not to be applicable if its condition is not satisfied. A policy component is not applicable if none of its children is applicable, or if its target is not satisfied.

As opposed to XACML, STAPL does not support `indeterminate` decisions. Instead, the authorization engine will yield an error that must be handled by the application. While this slightly reduces expressiveness of the policy language, it gravely simplifies the decision model.

Obligations. Policy components also support *obligations*. These operations must be performed whenever that component is found to be applicable.

Obligations can encompass anything from registering an audit log event, to showing a dialog to the user, and are not specified more closely. While obligations are an enabling functionality that supports specification of (among others) dynamic separation of duty policies, it will not be focused on in this dissertation.

Example policy. Consider an insurance company that uses the eDocs application introduced in Section 2.1 to share business documents such as invoices with smaller organizations, who may also use the platform. In this scenario, a lot of authorization constraints could be in place that need to be specified and enforced by the application. These constraints can involve multiple organizations that have their own authorization preferences.

Figure 2.7 lists a STAPL authorization policy that specifies such constraints for sales representatives employed by the insurance company. At the top level, the policy consists of three segments that involve (i) the authorization constraints of the insurance company ((1)), and (ii) a set of general constraints that are applied as a default decision ((2)). Realistically, this policy would likely encompass a lot more constraints, including those of the application itself, but for brevity we focus on a small subset of the policy.

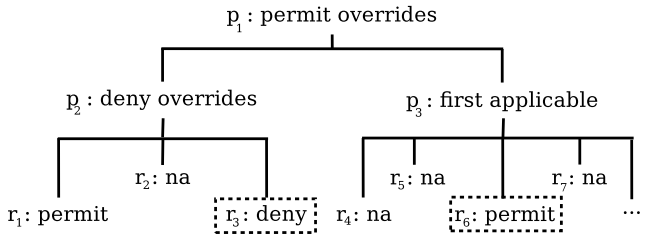


Figure 2.6 – Example evaluation of a STAPL policy. Locally decisive rules are indicated by a dotted border. For instance, for p_2 this is r_3 due to the `deny overrides` combining algorithm.

```
1  Policy("eDocs") := apply FirstApplicable to (
2    ...
3    Policy("insurance") := when (subject.organization === 1)   (1)
4                           apply DenyOverrides to (
5      Policy("sales") := when ("sales" in subject.roles &
6                               resource.type_ === "invoice")
7                         apply DenyOverrides to (
8        Policy("customers") :=
9              when (resource.owner in subject.customers)
10             apply FirstApplicable to (
11         Rule("read") := permit iff (action.id === "read"),
12         Rule("send") := permit iff (action.id === "send"),
13         Rule("other") := deny
14       ),
15       Rule("supervisor") := deny iff
16                          (resource.creator === subject.supervisor &
17                                  action.id === "send")
18     )
19   ),
20   ...
21   Rule("owner") := permit iff (action.id === "read" &        (2)
22                          resource.creator === subject.id),
23   Rule("default") := deny
24 )
```

Figure 2.7 – An example STAPL policy that restricts sales representatives of an insurance company, and describes default rules for a business document sharing platform. The eDocs policy component has no target expression, and hence is always applicable. Similarly, the default rule (at line 23) is always applicable because it has no condition.

At line 1, the eDocs policy is defined with a `first applicable` combining algorithm and no target expression. The latter indicates that any authorization request is applicable, and constitutes further evaluation of the policy's children. The insurance company policy at line 3 gives an example of a definition of a policy component with a target and combining algorithm. This component is a child element of the eDocs policy and is only applicable for subjects that have an attribute `subject.organization` of value `1`. Similarly, a rule is defined at line 21 that indicates that the `permit` decision must be propagated if its condition is satisfied (i.e., the reading subject is the `creator` of the resource).

The policy considers multiple attributes that are assigned to subject, resource, and action. For the subject, it assesses the `organization`, his/her `supervisor`, and a set of assigned customers (`customers`). These attributes are represented by unique identifiers associated with the subject and organizations. Moreover, a set of `roles` is evaluated for a subject (e.g., sales representative). For the resource, the policy considers the `type` of business document (e.g., invoices).

Also, it considers an identifier representing the subject that is the `creator`, and an identifier for the organization that is the `owner` of the resource. Both subjects and resources are recognized through a unique identifier attribute (`id`).

The application separates resource creation and sharing. When a resource is created, it is assigned an `owner` organization, which it will be shared with. However, it is only shared when a subject performs an explicit `send` operation.

The policy in Figure 2.7 generally restricts any subject from accessing any document they do not own (②). However, through the `first applicable` combining algorithm of the eDocs policy component, this constraint can be overridden. For instance, the insurance company supports sales representatives in reading and sending invoices of assigned customers. If that invoice was created by their supervisor, however, they cannot send it out.

Request	User		Resource		Action	
A	`id`	2	`type`	invoice	`id`	send
	`organization`	1	`owner`	2		
	`supervisor`	4	`creator`	4		
	`roles`	⟨ "sales" ⟩				
	`customers`	⟨ 2 ⟩				
B	`id`	3	`creator`	3	`id`	read
	`organization`	3				

Figure 2.8 – An authorization request. Conceptually, attribute references are substituted with their values during policy evaluation, which may involve retrieving additional required attributes from the PIP. For the policy of Figure 2.7, request A evaluates to `deny`, request B evaluates to `permit`.

To illustrate the decision process, consider the two authorization requests in Figure 2.8. These requests represent the context in which the policy is evaluated. The attribute references in the target and condition expressions in the policy components and rules, respectively, are substituted by the values of the authorization request. This enables an evaluation process that yields a decision that must be enforced by the application.

Authorization request A from Figure 2.8 also evaluates to `deny`. While the request concerns a sales operative of the insurance company that sends an invoice of an assigned customer, this document was created by the operative's supervisor which is explicitly restricted. For this request, both *insurance company* and *invoice sales* components are applicable, and the *send invoices* rule yields a `permit` decision. However, the *supervisor* rule also applies and overrides this. Due to the `deny overrides` algorithm of *insurance company*, this request yields a `deny` decision.

Finally, request B from Figure 2.8 yields a `permit` decision as it involves subjects reading a document created by them. The subject is not associated with an organization that has specified its own policy, and therefore the default rules of the eDocs policy are the only ones that apply. Thus, the *owner reads* rule is decisive for this request.

STAPL policies enable specification of fine-grained rules that assess attributes. The characteristics of the policy model enable leveraging its expressiveness for many purposes, which allows us to address the challenges posed in Table 2.1.

2.4 Key contributions

Section 2.1 discussed several motivating case studies for this dissertation. The resulting authorization constraints that were outlined in Table 2.1 are main drivers for the contributions that we will discuss in more detail in this section. While Section 2.1 focused on a small set of challenges according to the case studies, we found that these challenges are in fact more generally applicable. Nevertheless, Uzunov et al. [229] argued that no universal authorization model exists that can be suitable for all distributed, collaborative systems that take into account authorization for multiple organizations. We tend to agree with this argumentation, and emphasize that the contributions in this dissertation should be regarded as tools in the toolchain for authorization in multi-organizational systems, but in no way an ultimate solution for all problems that such systems may encounter.

Figure 2.9 illustrates how the contributions, packaged in a single the multi-organizational security middleware, can be deployed on a distributed, multi-tier application. Our primary focus is on applications that service user requests through thin web clients. The application performs calls to the access control middleware through its public APIs. The middleware can also be integrated with other access control responsibilities, such as federated authentication, which can be integrated with existing technologies such as OpenId [198]. Since this functionality does not perform any novel contributions to the state-of-the-art, it is not focused on in this thesis. The middleware abstracts from any operations that need to be performed in order to enforce authorization, including any network communication between distributed components and data access operations to attributes required for making an authorization decision. As such, it ensures a comprehensive authorization approach that can be supported in multi-tiered, multi-organizational applications. Note that the AMUSA and EBAC components could also be deployed in a more distributed fashion, which is not shown in the figure.

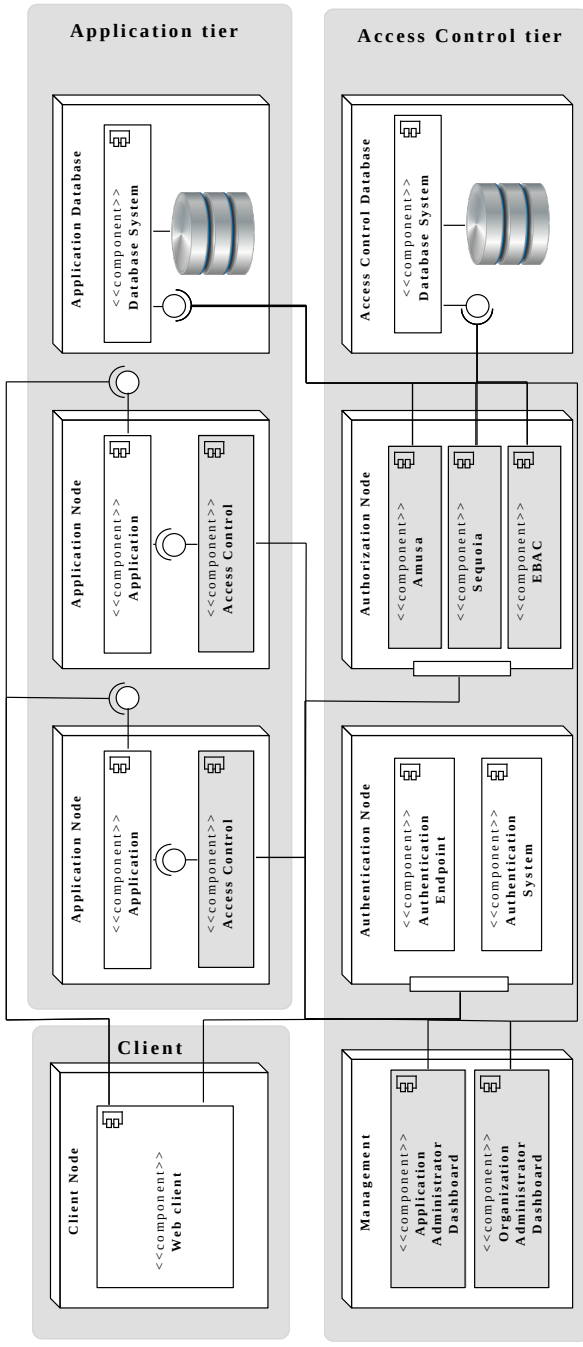


Figure 2.9 – The access control middleware can be deployed in support of a distributed, multi-tiered, multi-organizational application. The gray components are the focus of our contributions.

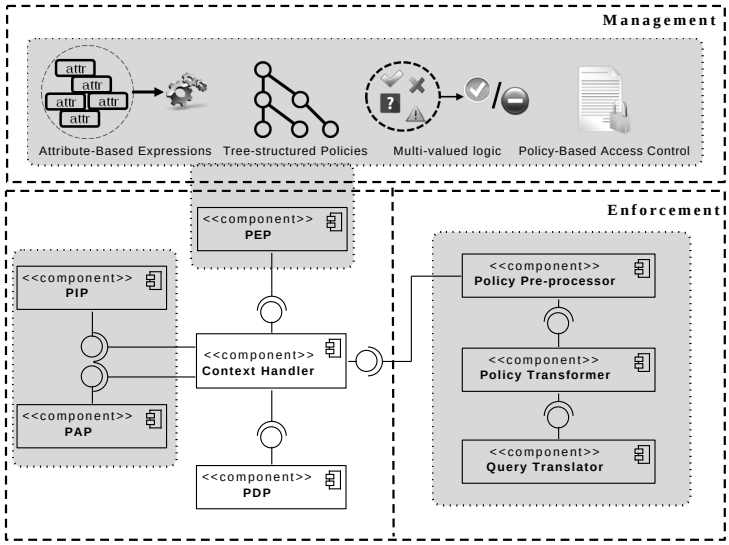


Figure 2.10 – Overview of the contributions of this dissertation, based on the background technologies discussed in this chapter. As part of this dissertation, we present Amusa (1), Sequoia (2), and Entity-Based Access Control (3).

The contributions of this dissertation are also presented conceptually in Figure 2.10. They are outlined in greater detail in Section 2.4.1–2.4.3, and discussed in-depth in the next chapters of this thesis. The figure illustrates two main areas of focus. First, management is performed through combining technologies such as tree-structured policies, attribute-based expressions, multi-valued logic, and policy-based access control into a management middleware called Amusa (①). Second, authorization enforcement is focused on through the traditional reference architecture presented in Figure 2.5, and through Sequoia, which supports authorization for data-focused operations (②). Lastly, the reference architecture is extended to support more expressive policies through Entity-Based Access Control (③).

The contributions in this dissertation are based on challenges that were drawn from case studies of multi-organizational systems. As such, they are primarily directed towards (but not limited to) providing an encapsulated approach for authorization that deals with multiple organizational entities. The contributions in this thesis assume a central authority that regulates access restrictions from separate administrative domains. Regulation of separate administrative domains is addressed through a mix of policy-based access control, attribute-based access control, and tree-structured policies. These technologies are leveraged in separate middleware layers that encapsulate authorization functionality and

provide collaboration support with a different emphasis for each contribution. Because of the separate administrative domains, this can be regarded as an amalgamate of mandatory policies with limited administrative delegation to security administrators of the involved organizations.

Chapter 1 briefly elaborated on the key contributions in this dissertation. By providing background in this chapter, this section now discusses them in more detail through the motivating authorization challenges in Table 2.1.

2.4.1 Amusa: authorization management and enforcement for multi-tenant SaaS applications

This contribution supports the management and enforcement of authorization policies of the service provider and the tenants in a context of multi-tenant Software-as-a-Service (SaaS) applications. The Amusa middleware supports specification of XACML and STAPL policies, that are combined with the authorization policies specified by other organizations through leveraging ABAC, PBAC and policy trees.

In particular, policy trees are leveraged to support management on three layers. First, the middleware itself outlines the basic concepts that are required for multi-tenant SaaS. It ties subjects to a tenant and defines tenant isolation constraints. Second, the service provider specifies policies to which all the tenants must adhere. Third, tenants specify policies that regard resources they have privileges for.

Through policy trees, these management levels can be organized in different policy components that are composed with combining algorithms preventing privilege escalations. Through attributes, that are also managed on three layers, subjects and resources are tied to an organization. This also enables flexible authorization specification for the involved tenants. Lastly, policy-based access control provides support for run-time customization of the authorization policies, as they are externalized from the application code and any changes made to the policies do not require implementation in the application code.

Amusa also provides an enforcement infrastructure that introduces an acceptable performance overhead. This infrastructure includes an interface for specifying policy enforcement points and hooking the decision engine with data sources for retrieving additional attributes necessary for policy evaluation. Configurable performance tactics also support overhead minimization that can be performed by the security administrators of the tenants.

While Amusa is focused on multi-tenant SaaS applications, the middleware

can support authorization in any multi-organizational system. The middleware serves as the central authority that enables organizations with different administrative domains to specify their own authorization preferences, and combines them in an authorization policy that supports collaborative sharing. As a result, AMUSA demonstrates that the combination of attribute-based access control, policy-based access control, and policy trees constitutes a promising approach for multi-organizational authorization systems.

This contribution also addresses two core challenges from Table 2.1. The multi-level management approach addresses challenge R1. Through enabling organizations to specify their own authorization policies, challenge R2 is also accommodated. Chapter 3 elaborates on the approach that supports this contribution, and provides an architecture, discussion, and extensive evaluation of the middleware.

2.4.2 Sequoia: query rewriting for authorization enforcement in data-driven applications

Policy-based access control provides several advantages. These were leveraged, together with other core technologies, to provide AMUSA as a general approach for authorization management and enforcement in multi-organizational applications. However, AMUSA is focused on control-focused applications where a single request typically yields a single authorization evaluation (e.g., rescheduling service appointments). On the other hand, *data-driven* applications are focused data-intensive operations such as search and data aggregation. Support for policy-based access control in such applications is generally lacking.

As a result, in order to leverage the advantages of policy-based access control for data-driven operations, a policy evaluation on each element that satisfies the constraints of the original query must be performed. For the attribute-based authorization policies supported by systems such as AMUSA, this can put an unacceptable strain on the performance [227].

The SEQUOIA middleware addresses this issue through the rewriting of queries based on dynamic run-time conditions. A translation of the authorization policy to a boolean expression is combined with the original query so that it returns only the results to which the user has privileges. As such, SEQUOIA enables policy-based management for data-driven applications and demonstrates that the application domain for AMUSA is not limited to control-focused applications. Instead, a single policy can restrict both control-focused as well as data-driven operations in a multi-organizational application. Through query rewriting,

Sequoia thus supports comprehensive enforcement of multi-organizational authorization policies.

In essence, Sequoia transforms STAPL policies to boolean expressions to achieve this. The authorization logic that is enveloped in policy trees, combining algorithms and attribute-based expressions is embedded in a single boolean expression that is than mapped onto the database structure and translated to a database-specific query that is combined with the original search or aggregation query. While state-of-the-art approaches can enforce organizational isolation through query composition in a domain-specific manner [16, 88], this approach enables full authorization policies to be embedded as part of the query.

From the motivating case studies, this contribution addresses challenge R3 of Table 2.1 by restricting search queries, but can apply the same approach for aggregation queries, also supporting challenge R4. Chapter 4 discusses the Sequoia middleware. It elaborates on the architecture and general approach of the middleware, and performs an extensive evaluation and discussion on the results.

2.4.3 EBAC: entity-based access control for multi-organizational authorization management

The last contribution of this dissertation involves a novel policy language and authorization system that improves the management of collaborative, multi-organizational applications. Entity-Based Access Control (EBAC) regards entities as first-class citizens of the expressions that specify the authorization constraints. These entities can involve both attributes and relationships, facilitating specification of rules that require existence of a relationship between two organizations and can evaluate attributes associated with the entities involved in the relationship.

EBAC provides a policy language that is based on STAPL and enables specification in terms of entities, based on the STAPL language introduced in Section 2.3. It also introduces an authorization engine for this policy language that introduces an acceptable performance overhead and a system to retrieve modeled entity attributes automatically.

As such, EBAC combines ReBAC and ABAC concepts into a unified policy model that is focused on entities instead of on relationships between them, or the attributes they hold. This facilitates multi-organizational authorization management, especially in a collaborative setting, as policies can now address the properties of the involved organizations and the users associated with them through relationships they hold. Through EBAC, policies can compare the

attributes of an organization that is associated with a resource or user, instead of requiring case-specific attributes that implicitly reflect such a relationship.

EBAC addresses challenge R5 from Table 2.1 by leveraging the support of relationships between entities. Challenge R6 is also supported, as EBAC supports specification of complex rules around recursive relationships, evaluating attributes associated with its entities along the path. Chapter 5 discusses the policy model and authorization system in more detail, and analyzes the characteristics and performance of the policy language and its evaluation.

Chapter 3

Amusa: authorization middleware for multi-tenant SaaS applications

This chapter presents Amusa, the first contribution of this dissertation. Amusa supports multi-organizational authorization through a multi-tenant authorization middleware that supports both management and enforcement of fine-grained policies. The Amusa middleware was first introduced in a publication at the ACM Symposium on Applied Computing [77], which served as a basis for this chapter.

The chapter provides an in-depth description of this middleware. Section 3.1 introduces the general problem and how it aligns with the challenges that we discussed in the previous chapters. This is expanded on in Section 3.2 that provides a key scenario and elicits a concrete set of challenges that are addressed by Amusa. Section 3.3 discusses the related work to Amusa in various research domains. Section 3.4 elaborates on architecture and core concepts of the Amusa middleware. Section 3.5 evaluates a prototype implementation of the middleware. Section 3.6 provides a general discussion of the middleware and analyzes advantages and drawbacks. Section 3.7 concludes this chapter.

3.1 Introduction

As indicated in the previous chapters, multi-organizational applications introduce several challenges with regard to access control. These include the support for self-management, isolation and collaboration, and run-time customization support.

These challenges are equivalent to those of Software as a Service (SaaS, [212]) applications, as these applications constitute a type of multi-organizational application. SaaS is a cloud-computing service model in which the software application is provided as an on-demand service to *tenants* (i.e., enterprise customer organizations) who are given access to a shared, typically web-based application hosted by a *service provider*. For tenants, the goal of employing SaaS is to outsource tasks associated with development of supporting software and the maintenance of infrastructure (e.g., servers, operating systems, storage), and focus on core responsibilities of the organization instead. For service providers, costs associated with maintenance and development can be shared among (enterprise) customers.

A central concept for access control in SaaS is that of *multi-tenancy*. In particular, *native* (application-level) multi-tenancy encompasses the sharing of a single application instance over various hosting resources among multiple tenants [212]. Native multi-tenancy enables the application provider to achieve a higher profit margin through leveraging the economies of scale, lowering operational costs at the expense of tenant isolation [107]. This implies that authorization measures must make sure that access is restricted between tenants based on resource ownership.

However, multi-tenancy also introduces challenges. This arises from the inherent complexity it has due to the shared multi-tenant architecture of the application, and the need for a persistent support for variability [29, 107, 135, 221] for each tenant through customization. This is especially true for security, and particularly for access control in the form of application-level access rules. For example, for the eDocs case study of Chapter 2, one tenant may wish to restrict sending invoices to only account managers that are assigned the intended customer. On the other hand, another tenant may prefer a restriction to sales managers during the office hours and only for customers of the European region.

Such challenges are not specific for multi-tenant SaaS applications, but apply to multi-organizational applications in general. Since the cloud service model is projected to gain popularity in the coming years [66], however, such challenges are put to the forefront even more. While this chapter focuses on authorization management and enforcement of multi-tenant SaaS applications, this contribution should be regarded as more generally applicable in the context

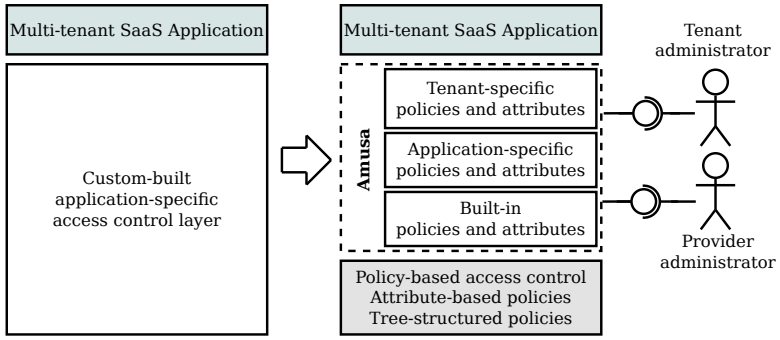


Figure 3.1 – Amusa provides a reusable middleware that supports configurability by both service provider and organizations in the context of SaaS applications. This is achieved by leveraging seminal technologies such as PBAC, ABAC, and tree-structured policies.

of multi-organizational applications.

The goal of this contribution is to address some of the key challenges with regard to multi-organizational authorization. In particular, Amusa supports run-time customizability of self-managed authorization policies by the tenant, while also composing these policies together securely with those of other tenants and the service provider. This contribution complies to fundamental concepts such as tenant isolation and collaborative sharing. As illustrated in Figure 3.1, Amusa encapsulates this complexity as part of a reusable middleware that supports both management and enforcement of access control at low performance overhead.

This chapter was based on a previous publication of the Amusa paper [77]. It will first analyze the challenges in more depth, followed by a discussion of related work. Next, it will introduce the Amusa middleware and evaluate a prototype. Lastly, we discuss some of the characteristics and conclude the chapter.

3.2 Authorization challenges to multi-tenancy

The Amusa middleware addresses requirements that were elicited from the electronic document processing and workforce management case studies from Chapter 2. These studies resulted in a key scenario that summarizes the requirements in the context of the eDocs case study.

The scenario involves two tenants, a large bank and a cable company, that use the document processing application to exchange business documents with their clients. However, access must be constrained on three levels:

1. The middleware must by default support tenant isolation so users of the cable company cannot access documents of the large bank.
2. The service provider (i.e., the eDocs SaaS provider) wants to restrict the document processing actions of the respective tenants according to their subscription plan. For example, the large bank can physically print out and send documents via snail mail using the application because it has a gold subscription, whereas the cable company can only do so digitally with its silver subscription plan.
3. The cable company wants to restrict its users to only read documents that are directed toward their assigned customers. On the other hand, the large bank wants to restrict access to certain documents to employees responsible for the European region, and only during office hours. Moreover, the bank wishes to relax tenant isolation for specific cases in order to be able to share documents with independent branch offices.

In order to support this, these preferences associated with different organizations should be combined in a way that the tenant that owns the document has control over its access specification, and by default the platform should assume that no access should be granted to other parties. Moreover, since this entails a SaaS platform with a potentially large set of tenants, it will have to ensure availability of the application. This means that further customization of these authorization rules should be supported *at run-time* in order to ensure that all tenants remain serviced at all times.

Requirements and goals

This key scenario already demonstrates key requirements for AMUSA. Generally, we focus on three goals for this contribution:

1. **Flexible authorization management.** The contribution must provide a flexible way to perform access control in a multi-organizational setting. In particular, AMUSA must (i) support the service provider to restrict tenants in terms of application-specific concepts, (ii) support tenants in constraining their users in terms of tenant-specific concepts, (iii) support tenants in making exceptions on the default isolation of resources for other tenants, enabling collaboration, (iv) combine all policies of the tenants and provider securely, and (v) dynamically bind and enforce applicable policies to each request to the application, enabling run-time customization.
2. **Configurable middleware.** Access control functionality should be encapsulated in a reusable and configurable middleware. Common functionality for multi-organizational applications in general, and SaaS applications in particular, should be built-in, including overridable tenant

isolation policies. Moreover, AMUSA should provide an authorization API that involves low instrumentation effort for the application developer.

3. **Efficiency.** The performance overhead introduced by the middleware should be as low as possible. This may involve support for performance tactics that can be adjusted for the application and tenants in order to improve performance.

The next section discusses how AMUSA is positioned with regard to related work. The remainder of this chapter will elaborate on how AMUSA addresses these requirements. The architecture builds on several core technologies that are combined in a directed way to achieve this. In particular, a combination of (i) policy-based access control, (ii) attribute-based access control, and (iii) tree-structured policies is leveraged into a SaaS-specific layer that enables secure and flexible management of multi-organizational policies. These technologies were discussed previously in Chapter 2, and are paramount to the architecture of AMUSA.

3.3 Related work

This work builds on an extensive body of research from various research domains, including multi-tenant cloud computing, grid systems, access control methods, and policy-based middleware.

First, multi-tenancy has driven quite some research. In this context, several access control models have been proposed. This has included extensions to role-based access control models [8, 223] and attribute-based models [186]. While these models provide a formal framework to reason about multi-tenant privilege management, they do not elaborate on how authorization rules are enforced, and do not provide the flexibility provided by AMUSA. The importance of such flexibility was already argued in 2007 by Guo et al. [107], when they identified high-level requirements to SaaS such as tenant isolation and support for customization to satisfy the needs of the tenants. The latter has also been argued in other works [29, 221]. While various works have been concerned with tenant isolation [60, 184], they do not support application-specific or tenant-specified exceptions to this rule. State of practice implementations of tenant isolation typically involve strict isolation in data stores. These implementations utilize tenant identifiers to differentiate the access by being concatenated in database queries (e.g., Google App Engine [88]). A similar approach has traditionally been adopted in research (e.g., Azeez et al. [16]). While this approach effectively separates tenant data, it leaves no room for easing the isolation for purposes of resource sharing.

Second, access control systems for grid computing have inspired Amusa. In grid computing, the focus for access control is primarily on scalable management for a potentially large set of *virtual organizations*, which may involve a large amount of users and resources [62]. This requires techniques that improve scalable management such as decoupling enforcement and policy evaluation. A similar approach is taken by Amusa, which also combines technologies such as attribute-based policies and policy trees into a configurable middleware.

Third, several works complement Amusa, as they are using similar building blocks for a different purpose. Fatema et al. [89] focus on privacy in multi-organizational systems and compose policies authored by multiple parties in a similar fashion to address privacy requirements. Similarly, these technologies are leveraged by Lazouski et al. [142] for usage control in Infrastructure as a Service (IaaS) systems. While the focus of these works differ, it would be interesting to see how they can complement Amusa.

Fourth, Amusa is inspired by works that support policy-driven middleware [40]. Sloman [214] applied policies for declarative management of access control in distributed systems. This has led to the definition of the Ponder policy language [73] which also enabled authorization specification and enforcement in distributed systems. Policy-based management has been applied in a large variety of domains, including for network management [110], managing of content-based publish/subscribe middleware [235], controlling information flow in multi-domain applications [17], or describing self-management behavior [139]. In general, a policy-based management approach is used to separate semantics from enforcement, which is also a goal for the Amusa middleware.

3.4 The Amusa middleware

In this section, we discuss the architecture and configuration capabilities of the Amusa middleware. In essence, Amusa can be summarized through four aspects:

1. **Three-layered management.** Management of authorization policies and the attributes they reason about occurs on three layers. These layers support management on the middleware-level, by the service provider, and by the tenants.
2. **Secure policy composition.** Policy management can occur on different layers, and tenants are empowered to specify their own authorization policies. This introduces security concerns with regard to composing these separate policies in a secure manner, which Amusa addresses through leveraging policy trees and combining algorithms.

3. **Multi-tiered enforcement architecture.** Besides access control management, the specified policies also need to be enforced. The middleware introduces a multi-tiered enforcement architecture that employs several performance tactics to decrease policy evaluation overhead.
4. **Development API.** Finally, the decisions produced by AMUSA need to be effectively enforced in the application it protects. In order to support this, AMUSA provides a low-effort development API that enables instrumentation of the application to work with the middleware.

The remainder of this section outlines these aspects in more detail. While AMUSA also provides support for authentication, we do not regard this as a contribution to our work and hence not focus on this aspect of the middleware.

3.4.1 Three-layered management

The middleware considers three parties to support management specifications. First, the *middleware itself* pre-defines common policies and attributes that are generally applicable across multi-organizational applications. These policies ensure default authorization guidelines. Second, the *provider* specifies policies and application-specific attributes that are focused on the application domain for which AMUSA is being employed. Because the provider presents the application, it has domain-specific knowledge about regulations and restrictions that need to be implemented. Implementation is achieved through enforcing policies that reason about tenants. Third, the *tenants* manage their own policies and attributes based on organization-specific concepts. This enables security administrators to perform self-management by customizing their own policies and the attributes of their users.

This three-layered model drives the management architecture of the middleware, and enables top-down authorization management support. Management of users is performed through attributes, which will impact their privileges based on the policy specification. We first regard management of resource and user attributes, and next elaborate on policy management in the three-layered model.

Attribute management

In order to favor reusability, AMUSA supports the gradual extension of the attribute set that can be utilized in expressions of the policy specification. Attribute management comprises the definition of attributes for users and resources on the one hand, and the assigning of values for these attributes for concrete users and resources on the other. As indicated previously, this

	Cable company	*Large bank*
Tenants	`user.assigned_customers`	`user.regions`
eDocs	`user.subscription_plan`, `resource.type`	
AMUSA	`user.id`, `user.tenant`	

Figure 3.2 – Three-layered attribute management enables the AMUSA middleware, the provider, and the tenants to define their own attributes, which can be evaluated in the policies.

is performed on three layers. Figure 3.2 provides an example of attributes definitions in these layers.

First, the middleware pre-defines attributes that are crucial for its correct functioning. For example, the unique identifiers associated with resources and users, as well as the tenant they are affiliated with are defined in this layer, and concrete values are assigned where possible. Other common, yet not vital attributes such as user roles and resource types are also defined on this level. Second, the provider may define and assign attributes to the resources in its application. Moreover, application-specific user attributes can also be assigned to map better to the application domain. For example, the eDocs application enables digital sharing of business documents, so an attribute may be defined by the provider to indicate the origin tenant for all resources. Since the enterprise customers of eDocs may entail independent branch offices that are associated with the large bank, the provider may also define a user attribute that indicates the tenant associated with the affiliate. Third, the tenants can define organization-specific user attributes. This enables them to reason about authorization more closely in terms of their organizational structure. For example, the large bank may specify an attribute that indicates the region(s) a user is responsible for. On the other hand, the cable company may want to add an attribute that represents the customers that are assigned to a certain user. These attribute definitions are depicted in Figure 3.2.

These attribute definitions and their assigned values remain separate for each tenant. However, they are allowed to use any attributes defined by the provider and the middleware. Some of the attributes are *immutable*, meaning that their values may not be reassigned at other layers. For example, the cable company cannot reassign the `tenant` attribute of a user to ensure the security properties of AMUSA that are discussed later in this section.

	Cable company	*Large bank*
Tenants	Deny if `resource.destination` not in `user.assigned_customers`	Override isolation if `user.tenant == 'branchA'` Deny if `'Europe'` not in `user.regions`
eDocs	Deny `action.id == 'post'` if `user.subscription_plan != 'gold'`	
AMUSA	Default tenant isolation policy	

Figure 3.3 – Three-layered policy management. AMUSA enables policy specification from the middleware, provider, and tenants.

Policy management

The management of attributes is an important part of authorization administration. However, support for policy specification is the real enabling factor for self-management. Similar to attribute management, this occurs on three layers. The policies defined on each of those layers are then composed securely, as will be discussed in the next section. Figure 3.3 provides an example of three-layered policy management.

We differentiate between conventional policies and collaboration policies. Conventional policies comprise authorization rules that are focused on constraining access further. Among others, this includes tenant policies that restrict their users. Collaboration policies are focused on *overriding* the default tenant isolation policy that restricts tenants from accessing each others' resources.

First, AMUSA itself specifies some built-in policies that indicate the default authorization behavior of the middleware. In particular, the default tenant isolation policy separates the resources so they can only be accessed among users of the owning tenants. This can be (limitedly) overridden at other layers through collaboration policies. The policies specified at this layer can only reason in terms of pre-defined attributes, as they are unaware what attributes will be defined at other layers. Second, providers specify policies that reason about tenant access. For example, the eDocs application may restrict certain operations to only tenants with a gold subscription plan. Similar to the AMUSA layer, these policies can only reason in terms of attributes defined by the provider, or alternatively, the AMUSA layer itself. Third, tenants specify policies reasoning about their own users, in terms of their own attributes or attributes from the other layers. For example, the large bank can restrict access to certain

resources to only employees of the European region if they access it during office hours.

The policies at these different layers need to be combined in order to make an access decision. The AMUSA middleware requires that none of the policies of the tenant, provider, or AMUSA itself to evaluate to a `deny` decision, and at least one of them to yields a `permit` decision, in order for an access request to be permitted.

One exception to this are the collaboration policies. In order to enable collaboration, both provider and tenants may override the default tenant isolation policy in a limited way. For example, the provider may loosen the isolation policy for certain types of documents when it comes to independent branch offices. Similarly, the large bank may explicitly grant access to invoices for a specific accounting organization.

3.4.2 Secure policy combination

In order to combine the policies as described in the previous section, AMUSA leverages the tree-structured policies and combining algorithms associated with the XACML policy language model.

How a policy is incorporated into the global authorization policy is based on its type. Conventional policies apply only to users affiliated with that tenant. For example, it does not make sense for a large bank to restrict users from the cable company, since such restrictions are normally already in place due to tenant isolation. The composition thus restricts applicability of conventional policies based on the `tenant` attribute. Collaboration policies specified by a certain tenant apply only if the accessed resource is owned by that tenant. Tenants can include additional constraints on both resources and users of the other tenant(s) the isolation is overridden for. For example, the cable company could indicate that all annual reports should be permitted for reading by any user of the application, as long as that report is no older than two years.

Figure 3.4 demonstrates the policy tree AMUSA uses to enforce these concepts. As seen in the figure, the collaboration policies constitute a separate sub-tree that propagates a `permit` decision if one of the collaboration policies yields that decision. When a tenant submits a collaboration policy, it is embedded in a policy component that restricts its applicability through a target expression. This expression imposes that the owner of the resource must be that tenant in order for said collaboration policy to apply. This prevents administrative privilege escalation by tenants. On the other hand, provider collaboration

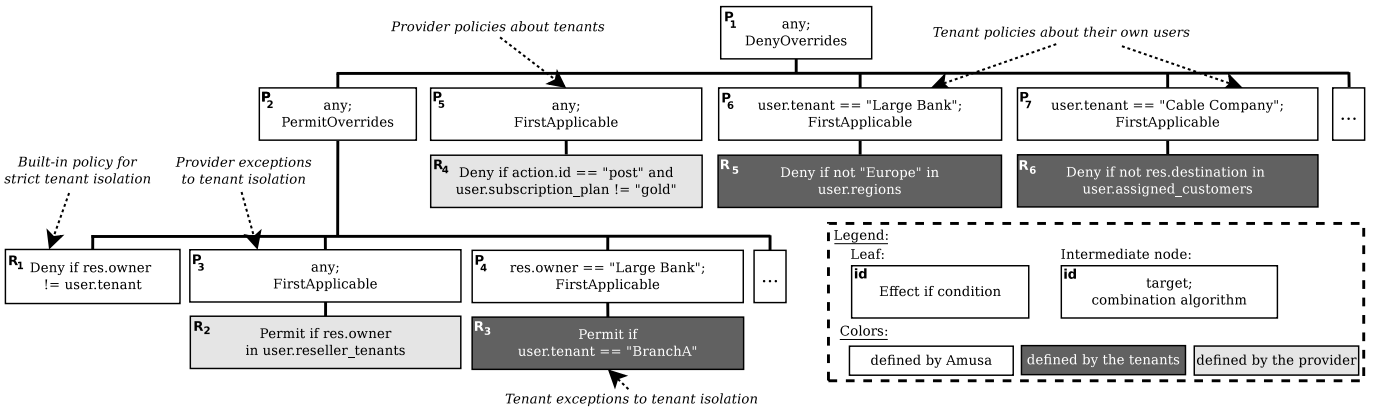


Figure 3.4 – An example of the secure policy composition for AMUSA. The policy tree combines the policies of all parties of the key scenario. The targets of individual policy components restrict their applicability and prevent administrative privilege escalation.

policies are not constrained by such a target and can therefore provide any type of exception of the default isolation rule R_1.

None of the collaboration policies (P_2), provider constraints (P_5), or the relevant tenant policy (e.g., P_6) may yield a `deny` decision, and at least one must evaluate to `permit`, in order to be given access according to the global AMUSA policy (P_1). Any conventional policies submitted by a tenant are encapsulated in a policy component that restricts applicability to only users of that tenant, in order to prevent escalation of administrative privileges. Note that if a user is affiliated with the large bank, policy P_7 will not be applicable and thus not issue a `deny` decision, but instead it is disregarded in the evaluation process altogether.

Due to the structure of the policy, the correct policies are dynamically bound at run-time, as only the appropriate policies will be applicable and require further evaluation. We identify six security properties that are guaranteed by the policy tree of Figure 3.4:

1. **Provider priority.** Tenants cannot override a deny decision originating from provider policies. This is achieved through the `deny overrides` combining algorithm of policy component P_1.
2. **Administrative scoping.** Tenants can only restrict access for their own users in conventional policies. This is ensured through encapsulating these policies in a policy component that only applies to their own users, as is the case with rule R_6 that is encapsulated in policy component P_7.
3. **Tenant binding.** Access requests only take into account policies of the appropriate tenant. This is a consequence of the *administrative scoping* property and the imposition of uniqueness and immutability of tenant identifiers in AMUSA.
4. **Enabled sharing.** The default tenant isolation policy can be overridden by both the provider and the tenants. This is achieved through the `permit overrides` combining algorithm in policy component P_2.
5. **Limited sharing.** Tenants can only specify exceptions to the default tenant isolation policy for resources they own. This is ensured by encapsulating the tenant's collaboration policy in a policy component that only applies to their own resources, as is the case for rule R_3 that is encapsulated in policy component P_4.
6. **Administrative isolation.** Tenants cannot revoke restrictions specified by other tenants on the latter's own resources. The middleware establishes this as a result of the previous security properties. As a result, any policy provided by a tenant on its own resources will not be overridable by other tenants, they can only restrict access further.

These security properties are all based on the principle of immutability of certain attributes. Some attributes defined and assigned by AMUSA (e.g., identifiers

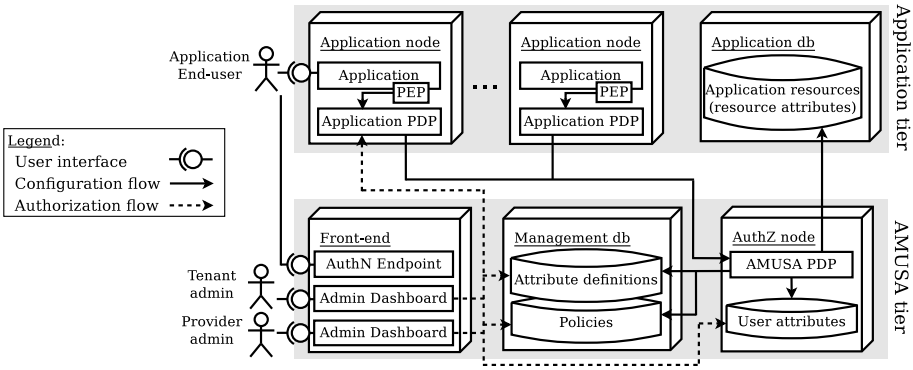


Figure 3.5 – The AMUSA architecture consists of two tiers, which support both physical and logical decoupling.

and tenant association of users and resources) or the provider (e.g., tenant subscription) cannot be reassigned by the tenants themselves. For example, a tenant cannot change the value assigned to a `user.tenant` attribute in order to isolate management capabilities.

To illustrate how the policy tree in Figure 3.4 enforces some of the security properties in practice, consider a user of the large bank that attempts to read an invoice owned by that bank. In this case, policy P_2 yields a `not applicable` result as none of its children, including R_1, have their conditions satisfied (although P_4 is applicable, its child rule is not). Moreover, the provider policy P_5 is not applicable here since it does not involve a `post` action. On the other hand, policy component P_6 does apply and hence the decision depends on whether the user is assigned the *"Europe"* region[1]. If the user was affiliated with *"BranchA"* instead, the action would be permitted due to the collaboration rule R_3, unless the branch specified further restrictions on its users.

3.4.3 Enforcement architecture

The enforcement architecture for AMUSA is based on the reference architecture for policy-based access control systems [232] that was previously introduced in Figure 2.5 of Chapter 2. However, this logical model is distributed and deployed in a way that enables reusability of the middleware and directed towards performance optimization.

[1]We assume that P_6 also includes a default `permit` rule specified by the tenant, that is not shown in Figure 3.4.

Figure 3.5 presents the Amusa architecture. The architecture consists of two tiers: an application tier and the Amusa tier. The application tier contains nodes that are focused primarily on application functionality, but also involves some authorization logic through PEP and Application PDP components. The Amusa tier holds nodes that support management, persistence, and evaluation of authorization policies and user attributes. The two tiers support both logical decoupling (which favors reusability) and physical decoupling (which favors performance through independent scaling). They enable multi-tiered policy evaluation motivated by configurable performance properties that optimize the evaluation process.

The application tier provides the SaaS application, and consists of a number of application nodes that host the application logic. Also, this tier contains the application database that persists the resources that must be protected by the middleware. Each application node also consists of a PEP that enforces the access decisions, and an Application PDP to evaluate a (partial) policy near application logic for performance optimization.

The Amusa tier provides the access control functionality. Through a front-end node, policies and attributes from both the provider and tenants are managed, and an authentication endpoint hooks into the authorization process. The definitions of attributes and the policies are stored on a database node. Lastly, an authorization node performs the actual policy evaluation through its Amusa PDP, and co-hosts the user attributes in order to leverage data locality for performance optimization. However, co-hosting the user attributes with the Amusa PDP is also a trade-off, as it introduces network communication overhead if the Application PDP contacts the authorization node because it lacks the user attributes required to perform the evaluation.

Authorization flow

The authorization flow in the architecture of Figure 3.5 starts from the PEP performing an access request to the local Application PDP. This request includes the attributes required to perform the policy evaluation. Hence, it at least comprises of the resource identifier, user identifier, and action, but may also include any additional attributes that will not need to be retrieved during the evaluation process. The Application PDP evaluates its part of the policy, and if no definite decision can be taken at this point, forwards the request to the Amusa PDP. The Amusa PDP evaluates the other part of the policy. These PDPs will issue requests to retrieve any missing attribute values dynamically if they are required for the policy evaluation. Either the Application PDP, or the

AMUSA PDP yields a decision that is returned to the PDP where it is enforced in the application.

Performance tactics

The architecture supports two tactics that can be configured to optimize the policy evaluation process, as policy evaluation may introduce a considerable overhead [227]. These tactics depend heavily on the policies and attributes that are used in the evaluation process, and should be configured by security experts per SaaS application and even per tenant to reduce overhead as much as possible.

Multi-tier PDP. Through support of both an Application PDP and AMUSA PDP, this architecture enables decomposing the policies in an optimal way in order to leverage the performance advantages they hold. On the one hand, the Application PDP is co-located with the application code itself, and can be leveraged to evaluate policies that require only attributes that are part of the access request in order to avoid network communication overhead. On the other hand, the AMUSA PDP is co-located with the user attributes, only requiring local database operations for its policy evaluation (unless additional resource attributes are required). Security experts should manually determine how the global policy should be distributed among these PDPs in order to optimize the evaluation process.

Pushing/pulling attributes. This performance tactic makes the trade-off between consistency and performance, as it involves the caching of attributes to avoid the overhead of attribute retrieval. For each user attribute, AMUSA supports configuring whether it should be *pushed* (i.e., included) in each access request. Alternatively, the user attribute could be *pulled* (i.e., retrieved) from the database during policy evaluation, incurring the overhead from the corresponding database operation. Caching can significantly reduce evaluation overhead. However, cached attributes also risk becoming stale, leading to incorrect access decisions if they are included in applicable authorization rules. Security experts can perform an assessment which attributes hold this particular risk (e.g., attributes reflecting the previous accesses to resources of a user) and which do not (e.g., roles) to fine-tune the policy evaluation process.

3.4.4 Application instrumentation

While AMUSA may provide valuable features for multi-tenant SaaS applications, it still requires some integration with the application code in order to yield

its benefits. The application developers thus require the proper tools to make use of this middleware without too much effort. With regard to authorization, application developers should (i) instrument the application with PEPs to request decisions from AMUSA, and (ii) extend AMUSA with attribute handlers for retrieving resource attributes from the application database.

The PEPs should be introduced at each point where authorization must be enforced. As a result, it is expected to be proportional to the application code. Therefore, the API that introduces the PEPs should be concise and straightforward to use.

```
1  // an application method
2  public Document viewDoc(Document doc, HttpSession session) {
3    User u = session.getUser();
4    Resource r = new Resource(doc.getId())
5      .addAttribute("type", "document")
6      .addAttribute("tenant", doc.getOwner());
7    Action a = new Action("view");
8    if(! pep.isAuthorized(s,r,a)) { return; }
9    ... // application logic
10 }
```

Figure 3.6 – Example of AMUSA instrumentation with the basic API. The API call supports inclusion of attributes in the access request. This must minimally include user and resource identifiers, and the performed action.

Figure 3.6 illustrates an example of how the authorization API can be employed. For both user and resource, attributes can be added according to the context of the access attempt, and the access request includes the attributes associated with the user, resource, action (and possibly environment attributes) that will be used in the policy evaluation process. The application developer should minimally include the identifiers of the relevant user and resource, and the action, but any other attributes can be included to optimize the evaluation process. The AMUSA middleware will perform the policy evaluation upon the `isAuthorized` call, and returns a boolean value reflecting the access decision.

```
1  @PreAuthorize("#doc", "view")
2  public Document viewDoc(Document doc, HttpSession session) {
3    ... // application logic
4  }
```

Figure 3.7 – Technology-specific approaches can be leveraged for more concise and simple authorization instrumentation.

This can be further simplified using technology-specific methods, such as Java Spring annotations [217]. The Java Spring application framework enables automatically injecting user session data into annotated controller methods with simple authorization statements. Such methods can be applied to further reduce implementation effort. An example of such a simplified API is provided in Figure 3.7. Other approaches to reduce instrumentation effort include `web.xml` configuration file support for Java web applications, aspect-oriented programming, or an application firewall.

```
1 interface AttributeHandler {
2   public boolean supports(AttrType t, String attrId);
3
4   public AttributeValue getValue(AttrType t, String attrId,
5         String resourceId);
6 }
```

Figure 3.8 – The attribute handler API must be implemented to support retrieving resource attributes in the the PDP.

Another responsibility of the application developers is to provide support for resource attribute retrieval from the application database. These attributes should be retrieved whenever they are required during the evaluation process, through a PDP that performs a request to the application database. In order to retain the reusability of Amusa, the application developer must implement the appropriate attribute handlers to support this. Figure 3.8 shows the interface for an attribute handler that the application developer must implement through database calls. Attribute handlers should be deployed as a separate component on the application nodes. This way, they represent PIPs that serve requests from the PDPs of the Amusa middleware.

3.5 Evaluation

The previous section motivated how Amusa addressed the requirements posed earlier in this chapter. This section demonstrates how Amusa does not introduce prohibitive overhead in terms of integration into the application, nor with regard to performance of the authorization.

3.5.1 Prototypes

In order to employ Amusa in a realistic context, we have developed a prototype implementation of the architecture in the previous section. Moreover, we

developed a prototype of the eDocs application [75] that supports the reading, sending, and management of documents.

The application prototype is a web application that focuses on a limited subset of functionality of its industry counterpart. It was implemented in Java, counts 1KLOC and uses Spring 3 Web MVC, Tomcat 7, and Hibernate. Documents are persisted in a MySQL database. This evaluation uses the scenario of a large bank similar to the one introduced in the key scenario.

The AMUSA prototype supports the management and enforcement infrastructure as described in the previous section. It was also written in Java and consists of 9.7KLOC. This does not include the XACML policy evaluation engine. An optimized version of the SunXACML engine was used to perform this task. The prototype communicates between nodes using Apache Thrift and Java RMI, and uses MySQL for attribute storage. Management interfaces are provided through a web front-end using Spring 3 Web MVC on Tomcat 7.

3.5.2 Development effort

Integration of AMUSA into the application comprises three tasks: (i) instrumenting authentication, (ii) instrumenting authorization through AMUSA access requests, and (iii) implementing attribute handlers to retrieve application resources from the database. We do not focus on the first task, as it does not entail a lot of effort and we do not regard it as a contribution.

Authorization instrumentation. Since authorization must be enforced on the operations that the application performs on resources, the effort required for instrumentation is expected to increase proportionally with the size of the application code. Our application prototype required authorization on our MVC controller methods that performed the creating, rendering, viewing, and deleting of documents. The instrumentation for these operations required only 57 LOC, which was 6.3% of the total application code. Instrumentation was performed using the basic API shown in Figure 3.6, so optimizations can driven this effort down even more. Consequently, this small and localized code illustrates the low instrumentation effort that is prone to optimizations.

Attribute handlers. In order to support resource attribute retrieval during the policy evaluation, AMUSA requires application-specific attribute handlers to be defined by the application developers. Alternatively, these attributes can also be pushed along with the access request to the middleware. In our scenario, no attributes were required as the alternative approach was sufficient to perform policy evaluation. Because our scenario is based on a realistic case study, this suggests that implementing such handlers may only be required in more complex

scenarios. Additionally, we expect the interface for the attribute handlers, provided in Figure 3.8, to be sufficiently simple to allow low development effort.

3.5.3 Performance overhead

While Amusa supports management and enforcement of authorization policies for multi-organizational applications, minimizing the performance overhead incurred by the middleware is an important concern. Hence, Amusa supports several architectural and configurable measures to reduce overhead of the policy evaluation. Evaluation overhead is a problem especially for XACML policies and has compelled a large body of work [146, 165, 168, 185, 227]. This part of the evaluation focuses primarily on the impact of the configurable performance tactics on the evaluation overhead. We first discuss the policies used in our tests, followed by an outline of the set-up. Next, we discuss our results.

Policy

The evaluation presented in this section was performed using a realistic policy based on the large bank as tenant of the eDocs SaaS application. This policy reasoned about the reading, sending, and deleting of business documents, and how access privileges were organized in the organizational structure of a large bank (e.g., its departments, offices, and roles).

The evaluation policy regards tenant isolation, and a provider-specified policy that restricts users from performing any operations if their tenant's credit is insufficient. Moreover, the policy includes tenant-specified rules. For example, one policy permits insurance agents to access documents if the corresponding customer is serviced by the agent's branch. Another policy enables secretaries to view inter-office communication files related to a project they are involved in. The resulting policy tree has a tree depth of 4, comprises 32 rules, and is described in 1119 lines of XACML.

This policy involves rules that compare attributes with each other or with concrete values. In total, the policy assesses 26 different attributes. This includes 12 user attributes such as the unique identifier, tenant, department, and roles, and 12 resource attributes that include the resource type, sender, destination, and topic. User attributes included 3 defined by Amusa, 2 defined by the provider, and 7 defined by the tenant. Moreover, an action attribute reflects the attempted operation, and an environment attribute represents the current time.

Table 3.1 – Test set-up description providing an overview of the run-time properties of 8 representative requests for the employed policy.

Request:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
#Tree nodes	10	9	23	23	32	22	28	33
↪#Rules	1	1	6	7	5	9	11	5
#Attributes	6	5	28	16	28	24	27	31
↪ #Unique	5	4	9	9	10	11	10	9
↪ #Resource	2	1	3	5	4	3	4	4
↪ #Subject	3	3	6	4	6	7	5	5
↪#Pushed	3	2	5	3	4	4	3	4

A XACML policy evaluation for a particular request does not necessarily regard every sub-policy and rule, and hence not every attribute. For each access request, this depends on the applicability of the sub-policies (if their target expressions are not satisfied, their child elements are not evaluated) and the combining algorithm (if a `permit` decision is encountered in a `permit overrides` policy, the remaining child elements do not require evaluation).

Table 3.1 summarizes, for 8 different representative access requests that were analyzed for the evaluation, the amount of policy elements, rules, and attributes that were assessed. This varies from requests that involve few policy elements and attributes (e.g., request 1), to requests that involve a lot of policy elements and attributes (e.g., request 8). The table shows the number of elements evaluated in the policy tree (can result into `not applicable`), the number of evaluated rules (leafs of the policy tree) and the number of attributes that are assessed to reach a decision. This total number of assessed attributes to reach a decision is further decomposed into the number of different required attributes (because the second request for the same attribute will be solved from the cache), the number of different resource attributes, the number of different user attributes and the number of these that are pushed. These numbers do not take into account identifiers of the user, the resource or the action.

Set-up

The evaluation regards a permutation of the two proposed configurable performance tactics, resulting in six cases. On the one hand, the evaluation regards the performance tactic that performed policy decomposition and

evaluation on different tiers. For this, we regarded (i) policy evaluation on only the Application PDP, (ii) only the AMUSA PDP, and (iii) a policy decomposition that requires evaluation on these two components. The decomposition in the latter case was based on whether the policy part was provider-specified (Application PDP) or tenant-specified (AMUSA PDP). Tenant-specified policies can be expected to evaluate user attributes more frequently, as they are primarily focused on the organizational constraints. On the other hand, the evaluation also regards the performance tactic of pushing or pulling all user attributes. Pushed user attributes are stored within the authenticated session. Resource attributes are always sent along with the access request.

For the tests, three different nodes were used containing (a) the client making the requests to the application, (b) the application logic, Application PDP, and database, and (c) the AMUSA PDP and user attribute database. The Application PDP was compiled as part of the application logic to reduce overhead.

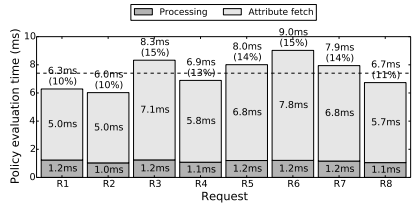
Results

The impact of the two configurable performance tactics are shown in Figure 3.9. This figure shows the evaluation times of the different access requests that are summarized in Table 3.1. This exposed several interesting properties.

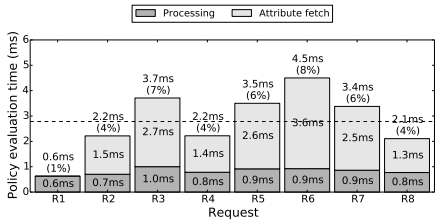
First, the basic case is represented by the Application PDP without any pushed attributes (Figure 3.9a). For this basic set-up, the retrieval of attributes takes up a lot of time with regard to the policy evaluation. These retrievals entail database operations that may involve network communication to the node hosting the user attribute database. On average, this amounts to 1.2ms of overhead per attribute fetch.

Second, the attribute pushing strategy has a significant impact on performance, as shown by comparing the overheads in Figure 3.9a and Figure 3.9b. Because the pushing of user attributes reduces the total amount of required attribute fetches, this also reduces the total overhead of the policy evaluation. However, this may introduce security issues related to inconsistency of attribute values, and should be enabled at the discretion of security experts.

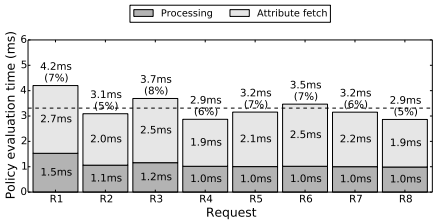
Third, whereas the first row demonstrates the overhead of the Application PDP deployment, a comparison with the second row (full AMUSA PDP policy deployment) shows that the attribute retrieval overhead is reduced by a factor of 2. This is caused by the absence of network communication for user attributes at the AMUSA PDP, as they are retrieved from a locally deployed database. The retrieval of user attributes locally amounts to about 0.6ms on average. However,



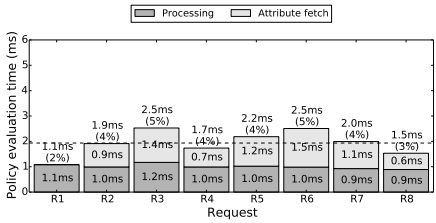
(a) Application PDP only, pulling user attributes.



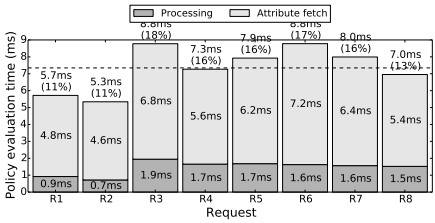
(b) Application PDP only, pushing user attributes.



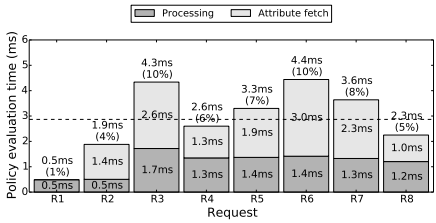
(c) Amusa PDP only, pulling user attributes.



(d) Amusa PDP only, pushing user attributes.



(e) Two-tiered PDP, pulling user attributes.



(f) Two-tiered PDP, pushing user attributes.

Figure 3.9 – The total policy evaluation time from the point of view of the PEP, for every request of Table 3.1 and for each of the six combinations of the two configurable performance tactics of Section 3.4. Each graph demonstrates the portion of the evaluation time spent on network overhead, retrieving attributes and processing the policy. Lower is better. The percentage on top of each bar represents the fraction of authorization overhead on the complete processing time of the application request. The dotted line represents the average over all requests for that set-up.

since the application node performs an access request to the AMUSA node, communication overhead is introduced for this remote communication that amounts to 1.25ms. As a result, especially with pushed user attributes this may disadvantage the AMUSA deployment approach as the remote communication may prove unnecessary to reduce database fetches. Whether or not a full AMUSA policy deployment is beneficial thus depends fully on the characteristics of both the access requests, policy, and the attributes that are pushed.

Fourth, the bottom row in Figure 3.9 reflects the combination of policy deployment partly on the Application PDP and AMUSA PDP. The first two requests demonstrate the impact for cases where the decision can be made solely by the Application PDP, as its decision any possible results yielded by the AMUSA PDP. However, other access requests actually involved a larger evaluation overhead as they required both attribute retrieval by the Application PDP and communication with the AMUSA PDP. This exposes the trade-off that must be made by security experts regarding how to decompose the policy and what attributes must be pushed on a per-case basis. In fact, this result shows that the current decomposition of the policy is sub-optimal. Existing techniques that automatically optimize policy decomposition may alleviate this issue [145].

Summary. Figure 3.9 demonstrates that the performance tactics may reduce the overhead, but depend heavily on the configuration and should be analyzed thoroughly by security experts. However, considering the overall results, on average AMUSA introduces only a low performance overhead that is acceptable for request-driven authorization in multi-tenant SaaS applications. While the best tactic with regard to performance in our scenario was to evaluate the full policy at the Application PDP with pushed user attributes, the policy decomposition, access requests, and how attributes are retrieved impose considerable influence on performance and should be assessed on a per-case basis by security experts.

Additional tests indicated that the amount of tenants did not significantly influence performance overhead, as their policies are separated in different policy components that are not evaluated further if the user does not belong to the corresponding tenant.

The optimal case leads to a mean overhead that is 2.8ms for all requests. With regard to the server-side application request time, this amounts to 4.9% of the server-side processing time. Considering the processing time of the client-side request, this comprises only 1.5%. We regard this as a low overhead, especially considering that these tests were performed for a non-optimized prototype.

3.6 Discussion

The evaluation results indicate that while the configurable performance tactics can be effective in reducing the overhead, they do require extensive domain- and security knowledge in order to maximize optimizations. Such knowledge must be achieved through pervasive monitoring of the policy evaluation process, the required attributes, and the incoming requests. Moreover, the frequency at which each attribute is changed for all users and what its security impact entails must be closely tracked. Consequently, the middleware must be equipped with proper monitoring capabilities to achieve this and present this in a clear way to the security administrators.

In the evaluation, we also briefly discussed the impact of the amount of tenants on the policy evaluation process. A growing number of tenants results in an increasing size of the policy tree. This was found not to impact performance significantly for at least up to 50 tenants. However, the small but existing overhead is expected to grow linearly with regard to the number of tenants, and hence for larger SaaS providers it may be beneficial to introduce more advanced policy primitives for matching the tenant policies dynamically (e.g., hash-based matching based on the tenant identifier). This was regarded as outside the scope of Amusa, but could be focused on in future work.

In general, however, the overall results demonstrated that the overhead for Amusa is low. It is not insignificant, however, and the impact of orthogonal performance optimization approaches such as more efficient attribute fetching [45], optimized evaluation engines [146], and policy refactoring [134] would constitute an interesting future investigation. This said, the Amusa architecture is focused on the characteristics of SaaS, which provides advantages especially with regard to its ability to scaling horizontally, and with the low overhead this demonstrates that performance should not be an impeding factor for its adoption.

Besides performance, the three-layered management approach of attributes and policies is focused on administrative scaling. The architecture presented this in three layers, but this may not be sufficient for large tenants that need to manage thousands of employees efficiently. In other words, support for *sub-tenants* and administrative delegation should be extended. Related work addresses this issue through administrative delegation models for role assignments [246], but does not support attribute-based access control and is hence not as flexible as Amusa. Two approaches to address this in Amusa are through (i) leveraging the sharing capabilities and regarding administrative entities as separate tenants that perform more granular authorization specification, or (ii) shifting to a multi-layered management entirely. The latter approach offers advantages in

user account mobility among sub-tenants, but it is yet unclear how this would affect policy composition and the security properties of AMUSA.

Related to this is the impact of multi-layered policies on the analysis of the reason a certain decision was yielded. Because a (sub-)tenant administrator is only *partly* privileged to manage the authorization policy, it may become complex to discover why a certain authorization result was decided (e.g., in case a provider policy yielded a `deny` decision). This is exacerbated in situations that the policies specified at higher layers are kept confidential. This issue is gaining attention in research (e.g., [71]) but it is yet unclear how it should be approached in a setting with multiple administrative domains with conflicting specifications, and where the motivation behind some of them may be classified.

3.7 Conclusion

This chapter presented AMUSA, a configurable middleware for efficient access control management of multi-tenant SaaS applications. Using state-of-the-art technologies such as policy-based access control, attribute-based access control, and tree-structured policies, AMUSA provides a flexible approach for security administrators of both the service provider and the tenants to specify their own authorization constraints, which are combined and enforced in a secure way.

The middleware supports both a management and enforcement infrastructure that is directed towards multi-tenant SaaS applications, and thus focuses on administrative scalability and efficient enforcement. In terms of management, the middleware does this through providing a three-layered attribute and policy management framework. In terms of enforcement, AMUSA provides a multi-tiered architecture that employs configurable performance tactics to minimize overhead. The evaluation shows that AMUSA requires only low integration effort. Moreover, the performance tactics can minimize performance overhead to an extent that it is low compared to the overall handling of server-side requests.

While AMUSA has been focused on SaaS applications, this middleware can be regarded as a contribution to multi-organizational applications in general. In essence, multi-tenant SaaS applications are a subset of multi-organizational applications that put a strong emphasis on organizational isolation, run-time customizability, and scalability with regard to requests. SaaS applications need to ensure near-continuous up-time and provide performance guarantees, which influenced architectural considerations for AMUSA. On the other hand, multi-tenant SaaS applications generally do not require support for collaborations between tenants. However, AMUSA does address this challenge as well, and hence

constitutes a contribution to multi-organizational authorization management and enforcement.

Chapter 4

Sequoia: scalable access control for data-driven operations

This chapter presents SEQUOIA, a data access middleware that enables attribute-based, application-level access control for data-focused operations in multi-tier, multi-organizational applications. Through query rewriting based on dynamic run-time conditions, SEQUOIA enforces external authorization policies on data-focused operations for both SQL and NoSQL database technologies. The middleware achieves this in a scalable manner with regard to the database size.

The chapter, which is based on several previous publications [36, 37, 39], is organized as follows: Section 4.1 introduces the problem and elaborates on how this corresponds with the challenges to multi-organizational authorization in general. Section 4.2 outlines the challenges that must be addressed by SEQUOIA and presents a key scenario. Section 4.3 places the contribution in the context of data access middleware and related work in database authorization. Section 4.4 presents the rewriting approach and discusses the architecture of SEQUOIA in greater detail. Section 4.5 evaluates a SEQUOIA prototype and examines the results. Section 4.6 provides a general discussion of the middleware and identifies some key future challenges. Section 4.7 concludes the chapter.

4.1 Introduction

Besides organizational isolation and support for sharing, run-time customization, low performance overhead, and separation of concerns are cornerstone requirements for authorization in multi-organizational applications. First, run-time customization enables policy modification without service interruption. Second, a low performance overhead ensures authorization enforcement does not impede the normal functioning of an application. Third, separation of concerns supports security management responsibilities to be delegated to security administrators of the involved organizations, which enables administrative scaling with regard to the enterprise customers of the application.

While Amusa demonstrated that these requirements can be addressed for control-focused operations (e.g., reading a document), several challenges remain in order to support them for data-focused operations such as search and data aggregation in multi-tier, multi-organizational applications. For example, the search results shown to a subject should only include data items that this subject is privileged to see. Similarly, when aggregating data according to categories, the statistics for each category should only include data items for which the acting subject is privileged, to avoid information leakage [195].

Current state-of-the art technologies fall short in properly addressing these requirements for data-focused operations. This applies to both authorization technologies directed to the data layer, as well as technologies of the application layer.

On the one hand, *database* authorization technologies, that are directed solely to the database tier in multi-tier architectures, are insufficient for several reasons. First, they require database administrators to be involved in policy specification. This violates the separation of concerns, and is complicated further due to an increasing heterogeneity of database systems. Worse, in the context of multi-organizational applications, this complicates self-management to a great extent as the administrators of the organizations may require access to the entire shared database for specifying their policies. Second, contemporary applications, especially large-scale deployments such as cloud applications, typically perform user management in the application. Consequently, they do not use the identity management system of the database. However, this is a prerequisite to leverage the database's access control system. Third, contemporary applications are generally designed according to a multi-tier architecture, and make use of privileged user accounts in the database to perform queries *on behalf of* a subject without the latter being further identified in the process. Consequently, access control is only performed for the entire application, instead of for individual subjects [196].

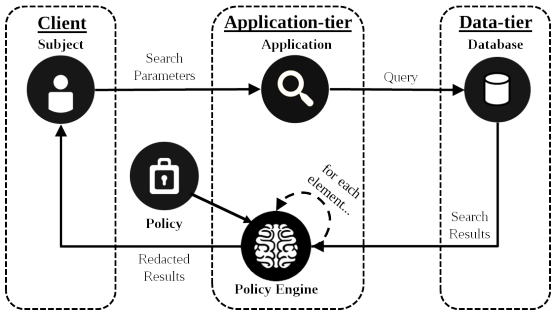


Figure 4.1 – The a posteriori filter approach evaluates an externalized policy for each data item of the result set of a query.

On the other hand, *application-level* access control approaches to this issue suffer from either poor rule specification granularity or poor performance. Tenant isolation approaches support organizational data partititioning through including organizational identifiers to constrain queries [60]. However, this domain-specific approach does not support flexible customization or sharing of data. An alternative approach combines the technologies of policy-based access control, attribute-based access control, and policy trees to perform authorization enforcement iteratively on each data item (i.e., *resource*) of the result set of a search or aggregation query. This approach, which we refer to as the *a posteriori filter*, is illustrated in Figure 4.1. The a posteriori filter could be used to support multi-organizational authorization, because it involves the evaluation of an externalized policy on each data item of the search result. However, this approach does not scale well with an increasing amount of search results, leading to prohibitive performance overhead for larger result sets.

Hence, there is a need for an approach that ensures support for requirements such as isolation, sharing, run-time customization, and separation of concerns, at a low authorization enforcement overhead. One approach that can be pursued is through performing run-time injection of the appropriate access rules into the search or data aggregation query based on the subject that performs the operation, as illustrated in Figure 4.2. This approach leverages the filtering system of the underlying database in order to select only data items for which the acting subject is privileged.

This chapter presents Sequoia, a reusable data access middleware that supports the rewriting approach, thereby providing a scalable authorization approach to support multi-organizational applications while taking into account the heterogeneity of database systems on which it is applied. Sequoia effectively provides authorization enforcement support for multi-organizational requirements such as isolation, sharing, run-time customization, and separation

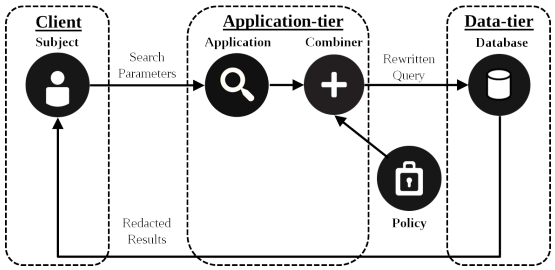


Figure 4.2 – The rewriting approach takes into account the authorization policy as part of the query.

of concerns for data-focused operations. Also, evaluation results demonstrate that the middleware introduces low performance overhead and scales well with regard to the size of the result set.

SEQUOIA ensures that the combination of policy-based access control, attribute-based access control, and policy trees can not only be used for control-focused applications, but also for data-driven operations. This ensures that the management approaches for multi-organizational authorization that were discussed in Chapter 3 can also be applied to ensure *comprehensive enforcement* of authorization policies. Hence, it constitutes an important contribution to multi-organizational authorization.

This chapter was based on a previous publications of SEQUOIA [36, 37, 39]. It first analyzes the challenges in more depth, followed by a discussion of the background and related work. Next, it introduces the SEQUOIA data access middleware and evaluates a prototype. Lastly, we discuss some of the characteristics and conclude the chapter.

4.2 Challenges

This section describes a key scenario that motivates the need for more comprehensive authorization enforcement support. This scenario is used as a running example throughout the remainder of the chapter. Next, this section elicits the requirements for the SEQUOIA middleware.

4.2.1 Key scenario

Sequoia was originally motivated by the electronic document processing case, the electronic workforce management case, and the managed security case from Chapter 2. These studies resulted in requirements that are described later in this section, and are first illustrated through a key scenario in the context of the eDocs case study.

Consider again a large bank that utilizes the eDocs application for managing invoices sent to their private customers. The bank shares all non-confidential invoices with a debt collector, which it trusts to have proper access controls in place. Additionally, since eDocs is a shared platform, the default isolation rule is applied and the platform restricts usage based on subscription plan.

The bank only manages invoices through the application, so its policy restricts operations to only this document type to prevent abuse. Moreover, some invoices are confidential as they involve very important private partners of the bank, and can only be operated on by members of the board directly. Normally, however, only members of the sales department can operate on invoices, and other users should be denied access.

Consider that an employee of the sales department wants to view all invoices that have not been paid yet. This could be supported through a search operation that filters based on this document property. However, sales members should not be permitted to see the invoices addressed to very important partners, so these should not be included in the search results.

Similarly, if an employee wants to determine the total amount of indebted credit over all invoices, the eDocs application can support this through data aggregation operations that calculate this sum inside the database. However, this should not include the outstanding debts of very important partners, as this may allow the employee to infer the indebted amount. Hence, the result should take into account organizational restrictions in its data-focused operations.

However, the same holds for any overarching rule that is specified by the provider. Figure 4.3 illustrates the authorization policy that includes both the authorization preferences of the bank, as well as a default isolation rule and authorization constraints of the eDocs provider, in a policy tree that resembles that of Amusa[1]. The provider grants itself the privilege to view any documents, and disables search operations for the bronze subscription plan. Moreover, collaboration only applies to non-confidential documents, so that users cannot (accidentally or maliciously) share confidential business documents with other organizations. The *view* operation comprises any action that enables

[1]For brevity, we omit policies of other organizations.

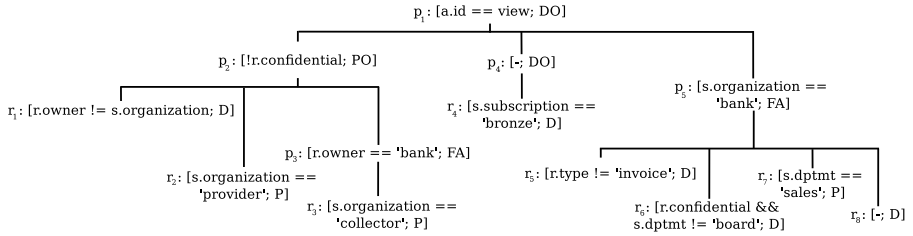


Figure 4.3 – Policy for the running example. We denote policy components (p_i) and rules (r_i) with their target expression and combining algorithm, or condition and effect, respectively. The policy includes `permit overrides` (PO), `deny overrides` (DO), and `first applicable` (FA) components and both rules with a `permit` (P) and `deny` (D) effect. Expressions evaluate attributes corresponding to subjects (s), resources (r), and actions (a).

inference of the existence of a document, be it a search operation or a data aggregation operation.

The policy depicted in Figure 4.3 can be enforced straightforwardly for control-focused applications. However, operations that involve large amounts of resources (i.e., *data-focused operations*) cannot enforce this as easily with a low performance overhead. In other words, in order to provide *comprehensive enforcement*, there is need for an approach that expands multi-organizational policy enforcement to the realm of database operations, while taking into account the requirements discussed in the scenario.

4.2.2 Requirements

Earlier work demonstrated that policy evaluation can yield a significant overhead, especially for expressive policy models such as XACML [147]. However, such policies should remain externalized from application and database, as this enables late specification [77] and just-in-time composition [28] of authorization restrictions which is essential to the administrative scalability of multi-organizational applications [221]. We identify four important requirements for SEQUOIA.

Multi-organizational support. The approach should provide support for key requirements in multi-organizational authorization such as support for isolation, sharing, run-time customization, and separation of concerns. For this, the focus should be on effective *enforcement* of authorization for supporting technologies, as the management capabilities were already addressed in AMUSA.

Support for heterogeneity of databases. An increasing popularity and

adoption of a variety of database systems implies that the approach must remain largely agnostic of how queries for the underlying database system are composed. Starting from a common expression that is translated to database-specific queries increases heterogeneity support as novel database systems can be introduced and supported more easily by Sequoia.

Development effort. The approach should require minimal integration effort. This implies that the authorization enforcement process should remain largely transparent to the application developer. Moreover, it should not impose any modification in the data structures that are managed by the database.

Performance. The approach should reduce the authorization evaluation overhead for data-focused operations as much as possible. With regard to the size of the data set, the proportional overhead increase should be limited as to ensure that the approach is feasible for larger databases.

The next section discusses some background concepts for the Sequoia middleware. Section 4.4 presents the architecture and outlines the rewriting approach that satisfies these requirements.

4.3 Background

This section provides a background to the concepts underlying to Sequoia and research related to this contribution.

Sequoia is focused on *multi-tier* architectures. Such architectures separate responsibilities related to the public API of an application, the application processing, and data management in different (physical or logical) layers. These are typically referred to as the presentation tier, business logic tier, and data tier, respectively. Multi-tier architectures support flexibility with regard to architectural decisions, facilitate reusability of the data tier among different applications, and enable scaling out each tier horizontally.

The data tier can hold any variety of storage systems. Over the years, many types of storage systems have emerged, leading to a common categorization into SQL and NoSQL systems. This section first provides a brief introduction for these storage systems. Next, we discuss the concept of data access middleware and how this can support versatility with regard to database communication. Finally, this section elaborates on the domain of authorization for data-focused applications, and regards database access control research relevant to our work.

4.3.1 Storage systems

Storage systems, and database systems in particular, have been the cornerstone of many software applications since the advent of computing. As a consequence, a large variety of systems have emerged over the last decades that are optimized for different goals. Notably, the last two decades have seen an emergence of database systems focused on horizontal scalability under the NoSQL umbrella.

A common categorization considers *relational databases* and *non-relational databases*. Relational databases have seen widespread success through imposing rigid, relational models upon the data that facilitate enforcing several measures on them, such as data-level constraints and ACID-compliant transactions. Non-relational databases can be *schemaless*, thereby providing more flexible data structures to the developers but also requiring some responsibilities normally handled by the storage system to be conducted by the application itself.

Relational databases comprise of tables[2] that retain data items with similar properties. For example, business documents can be stored in the same table as they hold the same properties such as type, sender, and content. These properties are stored in columns of the table, and may contain referrers to other tables (e.g., sender organizations), which can lead to elaborate *schemas* of relationships between data structures. Virtually every relational database use the standardized SQL query language [124] for performing operations (*queries*) on the data, and are hence commonly referred to as SQL databases.

Non-relational databases are generally more flexible, provide better horizontal scalability support, and higher availability guarantees than relational databases [109]. This comes at the cost of performance for certain data-focused operations, may reduce query expression capabilities, and may provide weaker consistency guarantees, depending on the particular database system in use. These *NoSQL* systems are traditionally categorized according to the data model they use in terms of *database models*. Popular database models include key-value stores (e.g., Redis [188]), wide-column stores (e.g., Cassandra [224]), and document stores (e.g., Elasticsearch [83]). Due to the application domain focus of these storage systems and limited support for boolean query expressions, our focus for Sequoia is on document stores.

Document stores are database systems that are focused on storage of semi-structured data [115]. These systems store all properties related to a data entity in a single instance called the *document*, not taking into account relationships between instances explicitly but rather leaving that as a responsibility of the application. This facilitates mapping of data into the database, and can speed

[2] Also referred to in literature as *relations*.

up development time, at the cost of data redundancy and consistency.

Terminology. There is some disparity in terminology for database systems. In this chapter, we assume resources to be represented as *rows* of (one or more) tables, or as separate *documents* in document stores. Similarly, attributes that correspond to the resources are assumed to be stored as the *cells* (or *fields*) of these rows (documents). These resources are referred to as *data items*. We expect that all resource attributes that are referred to in the policy are stored in the database, and a mapping is required to indicate how attributes can be translated onto columns (fields) in the resulting queries.

4.3.2 Data access middleware

Data access middleware constitutes the layer that interfaces between the application logic and the database system. This layer intercepts each request from the application to the database, and may perform transformations or other operations on it before forwarding it to the database.

As such, it provides an abstraction level that is typically used to cope with the variability of the underlying storage tier.

A wide variety of middleware systems support such abstractions, ranging from simple connectivity heterogeneity (e.g., JDBC [177]) to full-fledged object-relational mapping (e.g., JPA [178]) that enables application developers to reason in terms of application objects instead of how they are mapped by the database system. The increasing popularity of NoSQL systems has also driven polyglot object mapping middleware such as Kundera [122] to cope with the increased heterogeneity of database systems.

While these data access middleware systems are primarily focused on the functional abstractions of data-focused operations, they can also provide support for non-functional concerns such as security [128, 187] or performance [244]. As such, they support a configurable approach to cope with the variability of the underlying systems for various aspects of management.

4.3.3 Authorization for data-focused operations

Because databases are paramount in the functioning of businesses, they have attracted a lot of research attention with regard to security over the years [91]. This has primarily focused on relational databases, although NoSQL security is also attracting increased attention.

Our contribution is focused on restricting access to entire data entities (i.e., table *rows* and *documents*), and not their individual properties or data fields. Research commonly refers to this focus as *row-level access control* [26]. We envision queries from the application to be transparently modified to restrict privileges to data items, so that the access control system remains abstracted from the application developer. In research, this is termed the Truman model [195]. Our work does not elaborate on information flow technologies that are used in multi-level databases [152], as they require pervasive support from the database system and our contribution should abstract from this both for the application developer and the database.

We identify four general approaches that support some form of authorization on data-focused operations such as search or data aggregation queries on databases.

Fine-grained access control. This approach focuses on query rewriting techniques that leverage the access control system of the database to provide security [50, 97, 174, 195]. These techniques enforce rules specified in the native query language of the database, often achieved through the use of *views*. Generally, these techniques can provide scalable row-level access control [219].

However, fine-grained access control does have several issues. First, the rules are specified in the native query language and expect support from the access control system of the database. As a result, self-management in a multi-organizational setting is impeded. Moreover, this at least partly violates the principle of separation of concerns, as the database administrator needs to be involved in the policy specification process. Second, fine-grained access control generally assumes that subjects query the database directly under their own user account. However, contemporary applications generally adopt a multi-tier architecture that has the business logic tier perform queries to the data tier *on behalf of* subjects under a separate user account. As a consequence, subjects cannot be identified in the process [196, 197], and queries are thus performed on a per-application granularity level, not on a per-subject granularity level. Domain-specific technologies have emerged to alleviate this issue [64, 179, 195], but they tend to be database-specific, and still suffer from the other concerns with this approach. Third, large-scale deployments such as cloud applications typically do not manage the subjects in the identity-management system of the database. Instead, they are managed on an application-level or through external data sources. However, the access control system of the database requires these subjects to be identified through their own identity-management system, and cannot enforce the provisioned policies without this issue being resolved.

Sequoia solves these issues through external policies on the one hand, and substitution of subject properties as part of a rewritten query on the other.

In-database policy adjustments. This approach configures the database's access control system according to an external policy. Notable examples of this approach are SeSQLite [167] and MyABDAC [125].

SeSQLite extends the SQLite database to support SELinux [159] through database hooks. While this work supports separation of concerns, it requires database modifications. Moreover, SeSQLite focuses on lattice-based policies, which are not as fine-grained as the STAPL policies supported in Sequoia.

MyABDAC takes a similar approach by generating access control lists based on an external XACML policy. This enables the database to leverage the built-in access control system, yet still supports external policies. While MyABDAC supports separation of concerns and does not impede self-management, it suffers from scalability issues when externalized policies yield very large access control lists. Moreover, this approach requires the identity management system of the database to be used, and thus suffers similar issues as fine-grained access control approaches.

A posteriori filtering. In this approach, an externalized policy is evaluated against each data item in the result set. Bouncer [176] follows this approach for CPOL trust management systems. This approach supports policy-based access control and fine-grained policies. However, it introduces a considerable overhead especially when the result set size grows large. This is exaberated for fine-grained rules, such as those defined in XACML [227].

Since Sequoia leverages the filtering system of the database, it does not have this issue.

Query rewriting. Lastly, the approach pursued by Sequoia has also been adopted by several works.

Axiomatics Data Access Filter [15] in particular, rewrites externalized XACML policies as part of search queries for relational databases. However, they do not focus on the transformation process, and do not provide support for NoSQL systems. Longstaff et al. [148] did apply this approach for both SQL and NoSQL systems for TCM2 ABAC policies. Compared to these works, our contribution also provides a thorough evaluation that exposes the performance properties of the approach.

Colombo et al. have also recently taken this approach [63, 65]. First, they provide a work that enhances MongoDB with purpose-based access control policy enforcement [63]. They do this through a query rewriting technique that enhances the granularity level of the MongoDB policies. Second, the same authors apply a similar approach for document stores that support SQL++[175] for a custom ABAC policy model [65]. This work enables both row-level and

cell-level authorization. Compared to these works, our contribution evaluates a wider range of facets that expose the performance properties of the approach. Our extensible architecture also does not require compliance to SQL++ as it modularizes the translation to queries for each database system.

Orthogonal approaches. There are other security technologies orthogonal to our authorization approach that may be applied to harden security. In particular, homomorphic encryption [101] and secure query processing [4, 233] are technologies that enable data-level encryption and data-focused operations on encrypted databases. These technologies are complementary to our contribution, but also have challenges with regard to performance, and support for multi-layered, multi-organizational applications. Nonetheless, they are promising approaches to harden database security in a granular manner.

4.4 The Sequoia middleware

Sequoia is a data access middleware that intercepts requests to the underlying database and performs run-time injection of the relevant authorization rules together with the original search parameters in a query. The result of the query execution must contain only data items to which that subject has privileges. In other words, the query result contains only data items that satisfy both the original filter parameters and the authorization policy for the subject that performs the action.

In order to address the requirements from Section 4.2, Sequoia supports the combination of policy-based access control, attribute-based access control, and policy trees through enforcement of STAPL policies. These policies are transformed into boolean expressions that are then translated and composed with the originally provided search or aggregation query. The transformation must result in a semantically equivalent boolean expression. This is a non-trivial task as it involves dealing with concepts such as policy trees, multi-valued logic, and combining algorithms. This section describes the transformation approach in detail.

4.4.1 Overview

Figure 4.4 presents the general architecture of Sequoia. In order to come to the abovementioned query composition, the middleware performs (i) policy pre-processing, (ii) policy transformation, and (iii) translation to a query, respectively. Each of these steps is performed by a separate component.

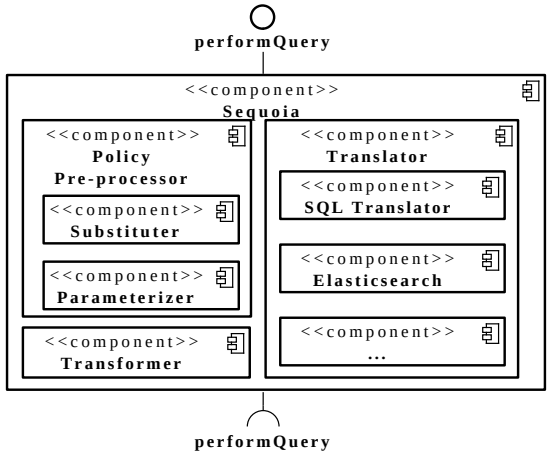


Figure 4.4 – SEQUOIA is realized as a data access middleware that receives data-focused requests from the application layer, performs transformations on them, and then forwards the transformed query to the data layer.

Throughout this process, original policy semantics are preserved. Hence, the result set for the rewriting approach is equivalent to the one returned to the subject if the policy is evaluated for each element that conforms to the query parameters. This comes down to the result set of the transformed query only containing the data items that yield a `permit` decision when a policy evaluation is performed on them for the same subject, action, and environment. Similarly, data aggregation must only take into account data items that yield a `permit` decision with a policy evaluation on its original query parameters. In other words, the result set of a transformed query should be *equivalent* to the result set of the original query on which a policy evaluation is performed for the acting subject.

In order to achieve this, the rewriting approach performs three steps, which are reflected in the components of the architecture depicted in Figure 4.4. First, the middleware performs policy pre-processing in which it *reduces* or *parameterizes* the policy that must be enforced according to the acting subject and attempted action. Second, the resulting policy is *transformed* to a (generic) boolean expression that returns `true` only if the policy evaluation would yield a `permit` decision for each data item (i.e., resource) of a data set. Third, the boolean expression is *translated* and *composed* with the original query parameters according to the underlying database. This composed query is then forwarded to the database.

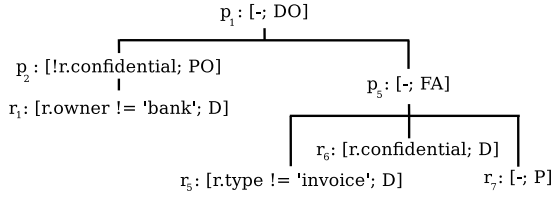


Figure 4.5 – Result of minimizing the policy from Figure 4.3 for a subject with `organization=bank`, `subscription=gold` and `dptmt=sales`, and a `view` action. This yields omission of rules r_2, r_4, and r_8, and policy components p_3 and p_4.

4.4.2 Policy pre-processing

The *policy pre-processor* component distinguishes between two approaches that make a trade-off between performance on the one hand, and security on the other. To prioritize performance, the middleware can perform *partial substitution* and evaluation of all expressions of the policy according to all attributes of the subject, action, and environment, in order to minimize the policy. Alternatively, security can be prioritized through *parameterization* of non-resource attributes that are compared in the policy expressions so that they can be translated and substituted later in a parameterized query.

Partial substitution

In this approach, all non-resource attributes are substituted by concrete values according to the request context (i.e., the relevant subject, action, and environment) in every expression of the policy. The policy adapter first retrieves all values of non-resource attributes referred to in the policy from the relevant data sources, such as from directory services, session parameters, or application properties. Similar to Rissanen [194], boolean logic and combining algorithms can then be leveraged by a *partial evaluation engine* to minimize the policy in a way that it still reflects the transformation context that must be applied to each data item of a query.

An example of a minimized policy is given in Figure 4.5. This figure demonstrates how the policy introduced in Figure 4.3 is reduced for a `view` action and a subject with `organization=bank`, `subscription=gold`, and `dptmt=sales`. The resulting policy is reduced by four policy elements and six expressions.

The key insight for partial substitution is that in the context of a single data-focused operation, the same query, composed of original search parameters and the transformed policy, is performed on many data items for a single subject,

action, and environment. Substitution of non-resource attributes thus enables pruning of (i) expressions of targets and conditions, and (ii) the policy tree.

First, substitution of non-resource attributes with concrete values may enable the pre-processor to determine whether an expression is always or never satisfied. For example, consider the target of policy component p_1 of Figure 4.3 with a transformation context that entails the `view` action. Substitution shows that the target of p_1 is always applicable and can thus be omitted altogether. Similarly, the condition of r_6 can be partially omitted for subjects of the sales department, thus reducing the policy size and eventually resulting in a smaller query.

Second, combining algorithms and policy element applicability can also be leveraged to minimize the authorization policy. Policy elements for which the condition or target is never satisfied will never be applicable, and can thus be omitted from the policy for the transformation context. Moreover, policy elements that are always applicable enable leveraging combining algorithm logic. For example, consider p_5 of Figure 4.3 a `first applicable` policy component. If the transformation context has a subject with `dptmt=sales`, rule r_8 can be removed for the minimized policy as the `permit` decision of r_7, which is always applicable in this context, will always be prioritized over r_8.

Partial substitution must be repeated for each different transformation context (i.e., different set of subject, action, or environment attributes), as different subjects may have different permissions due to their attributes, and a difference in environment (e.g., time of day) may affect the privileges for the data-focused operations. However, this substitution can also lead to significant policy reductions, yielding smaller queries and hence a lower query execution overhead. Our case studies seem to confirm these assumptions, yielding reductions of about 72% up to 92% of the policy elements.

Parameterization

Sequoia can also leverage the query parameterization functionality that is supported by some database systems. In this approach, instead of substituting non-resource attributes with concrete values in the policy, the middleware *parameterizes* them so they can be filled out just-in-time.

To achieve this, the middleware records the sequence of the assessed attributes in the policy and keeps track of this in the transformation step itself, so that the query can be stored and reused by different subjects that perform the same data-focused operation. Whenever a subject wants to perform this operation, the middleware forwards the attribute values associated with that subject to the database system, which uses them as parameters in the requested query.

In relational databases, parameterized queries are supported under the concept of *prepared statements* that are stored by the database system and executed with different parameter values. This is a common countermeasure for query injection attacks [234], which escapes the intended flow of a query to perform operations the acting subject is not privileged for. Using query parameterization, input validation on the query parameters (i.e., the attribute values) is not required as the flow of the query is fixed and cannot be escaped by the parameterized values.

At the cost of a larger query and thus increased performance overhead, SEQUOIA supports query parameterization to leverage this security measure.

4.4.3 Transformation to boolean expressions

Given a minimized or parameterized STAPL policy, the *transformer* component converts the authorization constraints to a boolean expression. This involves handling concepts such as policy trees, combining algorithms, and multi-valued logic. These concepts complicate the transformation process to a great extent, and each of these need to be addressed in order to support the query rewriting approach.

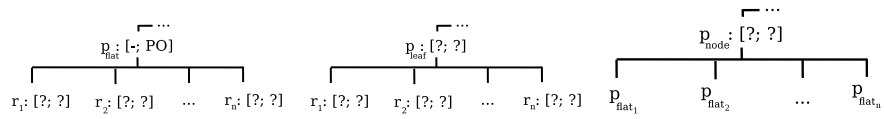
In case the substitution approach was followed, the resulting boolean expression will compare resource attributes with each other and with concrete values. Non-resource attributes are factored out at this point as they were substituted with concrete values. Similarly, the parameterization approach yields a policy that compares resource attributes with each other or with placeholders for parameter values.

In order to ensure that the authorization policy is enforced in the same way as in a traditional policy evaluation scenario, we need to ensure that the resulting boolean expression reflects the original policy in a equivalent manner.

Equivalence. The transformation result retains equivalence over `permit` decisions in the policy evaluation. In other words, when a resource is permitted under a policy evaluation with the same subject, action, and environment that was used as a basis for the transformation, the boolean expression is satisfied if evaluated. In all other cases, the boolean expression evaluates to `false`. This also filters out any data items for which the evaluation is indecisive (i.e., yields `not applicable`). Evaluation errors should lead to the query being aborted altogether. This does not fully conform to the XACML 3.0 specification [172], which introduces indeterminate decisions that can still lead to permitting or denying the access attempt in certain situations (e.g., if an attribute could not be retrieved). However, whereas such specifications may apply in control-focused

applications, our use cases did not indicate a need for such exceptional behavior. Moreover, evaluation errors can not always be handled gracefully for individual data items.

Transformation process. The goal of the transformation process is to extract a boolean expression out of a policy consisting of a single `permit overrides` policy component with no target and only rules as children, also called a *flat component*. This section outlines the process of the transformation. For an in-depth analysis and description of the transformation functions according to the combining algorithms, we refer to [38].



(a) Flat component **(b)** Leaf component **(c)** Node component

Figure 4.6 – The transformation process considers three types of policy components. Flat components have a `permit overrides algorithm`, no target, and only rules as children. Leaf components have only rules as children. Node components have flat components as children.

In order to extract a boolean expression out of a flat component, the transformation process iterates over two steps that gradually flatten the policy tree. The two steps consider *leaf components* and *node components*, respectively. Leaf components are policy components that contain only rules as children. In contrast, node components have flat components as their children. These components are illustrated in Figure 4.6. The transformation process is illustrated in Figure 4.7.

The **first step**, depicted in Figure 4.8, converts all leaf components to an equivalent flat component. In other words, every component with only rules as children is transformed to an equivalent policy component with `permit overrides` algorithm and no target. This is done through combining algorithm-specific conversions and conjunction of every condition of the child rules with the target expression of their parent. Figure 4.9 shows an example of this step that is applied to the minimized policy of Figure 4.5. As the figure shows, policy components p_2 and p_5 are transformed to flat components.

The **second step**, depicted in Figure 4.10 regards every *node component* that only has flat components as children. The rules of each of these flat components are pulled up into the grandparent component and the node component is transformed to a `permit overrides` combining algorithm. Semantic equivalence is retained through transformation methods that are

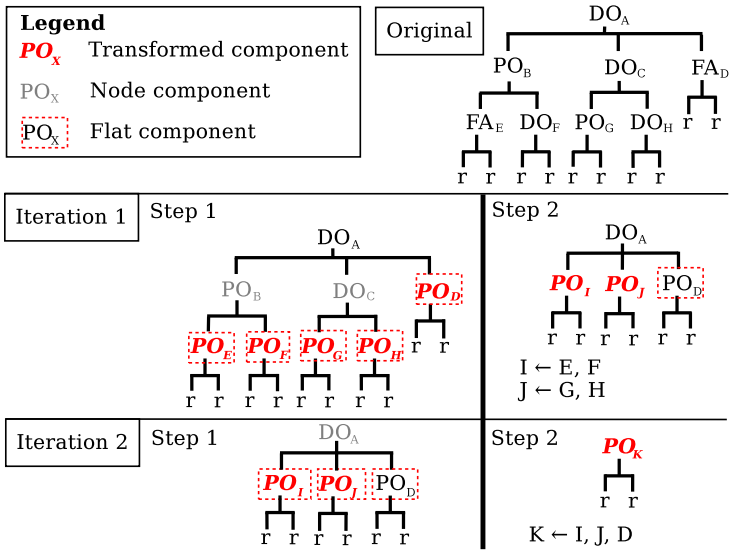


Figure 4.7 – The transformation process gradually flattens the policy until it yields a single flat component. From this, it can extract the boolean expression.

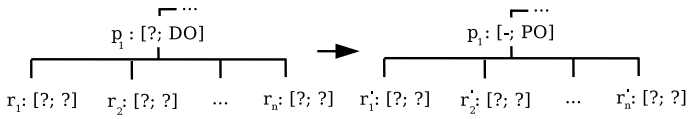


Figure 4.8 – The first step considers only leaf components. It transforms them into equivalent flat components.

established by the combining algorithm of the node component, which determine how the rules are pulled up to that component as well. Figure 4.11 demonstrates the result of applying the second step on p_1 of Figure 4.9. The rules of p'_2 and p'_5 are adopted in p'_1, resulting in a single permit rule r_Ψ and deny rules r'_1, r_5, and r_6. Note that resulting policy component p'_1 is a flat component from which the boolean expression can be extracted.

By iterating over these steps, the policy tree can be gradually flattened to a single *flat* component that has a `permit overrides` combining algorithm, no target, and contains only rules as children. A semantic equivalent boolean expression of the original policy can then be extracted through a disjunction of the conditions of all `permit` rules. For the example in Figure 4.11, the resulting boolean expression thus becomes

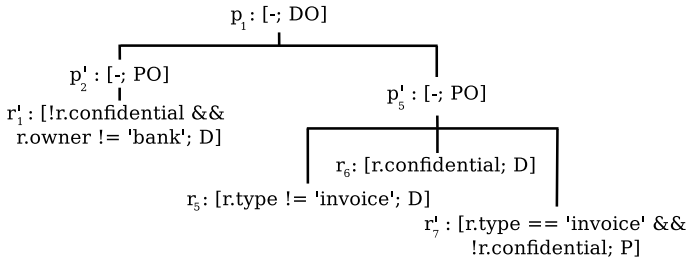


Figure 4.9 – The first transformation step applied to the minimized policy of Figure 4.5. Policy components p_2 and p_5 are transformed to equivalent flat components p'_2 and p'_5, respectively.

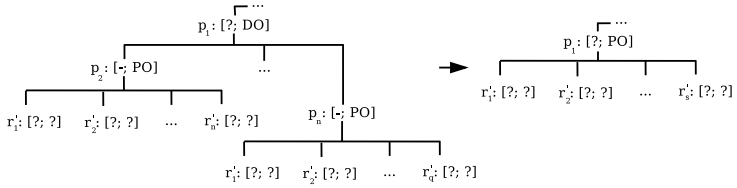


Figure 4.10 – The second step considers only node components. It adopts the rules of its child components.

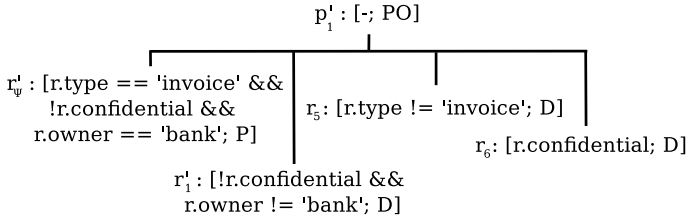


Figure 4.11 – The second transformation step applied to the node component of Figure 4.9.

$$\texttt{r.type == "invoice"} \land \neg\texttt{r.confidential} \land \texttt{r.owner == "bank"}$$

The `r.owner == "bank"` segment of the expression stems from the transformation of policy component p_2. The other two segments stem from rules r_5 and r_6 that were originally specified under p_5.

When translated to a query, this expression imposes filtering properties on the data items which they must satisfy in order to comply to the authorization policy. It is only satisfied for data items to which the subject that performs the query has privileges, and thus effectively enables enforcing authorization policies in search and aggregation queries.

The transformation process is driven by the combining algorithms of the policy components. These algorithms determine how rules can be converted and adopted in both transformation steps. For brevity, we do not elaborate on this here. Instead, we refer to [38] for an in-depth analysis.

4.4.4 Translation

The boolean expression resulting from the transformation is database-agnostic. This offers two advantages. First, it enables support for multiple database systems and query language dialects. Second, the resource attributes referred to in the boolean expression can more flexibly be mapped on the underlying data model. The middleware can translate this mapping of attributes to fields of data rows and documents on the fly. Hence, based on a boolean expression, the middleware can generate a query clause that is tailored to the underlying database and data model. This is performed by the *translator* component.

The responsibility of the translator component is threefold. First, it maps each resource attribute onto the data model of the data set on which the query must be executed. Second, the component performs a translation of the boolean expression to the syntax of the query language of the targeted database and composes the result with the original search parameters provided by the subject. Third, it executes the resulting composed query on the database. In case of parameterization, the component performs this execution with an ordered list of the appropriate attribute values as parameters.

In order to address the second responsibility, the component depends on database-specific modules that will handle the translations of the boolean expressions and data model mappings to the query language supported by that database. For example, an Elasticsearch [83] module will generate a JSON query that can be executed by an Elasticsearch database node that complies to the provided data model.

A security administrator must provide a mapping when configuring SEQUOIA to support a certain database. This will lead to a translation of attributes to data fields, but may introduce complexities, especially for relational databases. Because relational databases do not always manage the data fields associated with a data item in a single table, the translated query may require *joins* that associate multiple tables with each other in order to assess these fields as part of the query, in order to fully conform to the authorization policy. This not only complicates the mapping configuration by the security administrator, but may also lead to more extensive and complex queries. However, our case studies suggest that the amount of required joins is typically limited, and hence does not introduce significant overhead.

4.5 Evaluation

This section analyzes the performance properties of the rewriting approach that is central to SEQUOIA. We do this through evaluating a prototype implementation based on the architecture presented in Section 4.4. This prototype supports enforcement of STAPL policies on two types of storage systems. First, the prototype supports MariaDB [225], a relational database based on the MySQL database system. Second, it also supports Elasticsearch [83], a NoSQL document store and search engine that is ranked among the ten most popular database systems[3]. The choice of these database systems was motivated by our case studies.

The evaluation assesses several aspects of the rewriting approach for both systems and compares it with the *a posteriori filter* approach. The a posteriori filter was illustrated in Figure 4.1. It performs policy evaluation on each data item of the result set to determine what items should be included in the final result. Because this approach makes use of an externalized policy, it can address the concerns of separation of concerns and run-time customization, without requiring the subject to be identified in the database access control system.

Setup. The evaluation assumes a multi-tier architecture that consists of two nodes, (i) the database node, and (ii) the client node. First, a *database node* hosts the database system and returns the search result set to the requester. The database systems considered in this evaluation were MariaDB 10.1 and Elasticsearch 5.4. Second, a *client node* reflects the application that performs data requests. This node issues the search and data aggregation queries. Moreover, this node performs either the query rewriting (done by SEQUOIA), or applies the a posteriori filter method on the result set it retrieves from the

[3]According to the DB engines ranking, see `https://db-engines.com/en/ranking`.

database node. Both nodes were deployed on a shared cloud platform. The nodes hosted a Fedora 24 operating system, with an Intel Xeon E5-2680 CPU and 8GB of internal memory.

Each test was repeated 1000 times after 100 warmups. The caching on both MariaDB and Elasticsearch was disabled as much as possible. The client node communicated with MariaDB through JDBC, serializing all incoming data. For Elasticsearch, the native Java API [84] was used for communication, with the scroll size configuration maximized to 10,000 data items per call to minimize the communication overhead between the client node and the database node.

4.5.1 Overview

The evaluation exposed several performance properties of Sequoia. In order to determine them, we have performed six tests:

1. **Scalability.** This test evaluates the scalability of the rewriting approach for an increasing data set size, and compared it with the execution performance of a basic aggregation query with which the policy was combined.
2. **Pre-processing and a posteriori variations.** This test regards the performance overhead associated with variations of both the a posteriori filter and rewriting approach for a search query on an increasing data set.
3. **Overhead analysis.** This test analyzes the impact of several aspects of the rewriting approach, such as transformation overhead and query processing overhead, on the overall performance.
4. **Policy complexity.** This test assessed the impact of complex policies on performance. In particular, it evaluated the impact of the policy tree size and combining algorithms on the overhead.
5. **Attribute impact.** This test evaluated the performance impact of an increasing amount of attributes that must be considered by the query processor and policy evaluation engine.
6. **Proportion of entitlement.** This test analyzes the performance impact of the proportion of data items of the entire data set that a subject is privileged to.

Most tests were performed for both MariaDB and Elasticsearch. For brevity, this section only presents the results of one database system for each test, and discusses any notable differences whenever they occurred. Unless stated otherwise, we compared the results for the a posteriori filter and rewriting approach in which partial substitution first minimized the policy.

Policy and data set. For the first three tests, we performed the evaluation

based on the electronic document processing case study [75] that was introduced in Chapter 2. For this, we used the same authorization policy that was utilized for the evaluation of AMUSA, albeit that this policy was first translated to STAPL. The policy consists of 32 policy components and 63 rules that regard 26 different attributes. Resource attributes were included as fields of the data entities in the database.

The databases stored large sets of data items with various properties such as the owner, organization, destination, and topic. The evaluation results regard a subject that is entitled to about 40% of the data set, regardless of the size of the data set. This is referred to as the *proportion of entitlement*.

The latter three tests use artificially generated policies and data set that expose performance properties. The individual tests elaborate on how the respective policies and data sets were generated.

4.5.2 Scalability

This test assesses the scalability of the rewriting approach with an increasing data set size for a data aggregation operation. The results for Elasticsearch are shown in Figure 4.12. The figure shows that the filtering that is performed for enforcing the authorization policy impacts performance compared to the same query on the entire data set, which does not perform any authorization. However, for larger data sets (i.e., 100,000 data items), the rewriting approach actually performs better than the aggregation query on the entire data set,

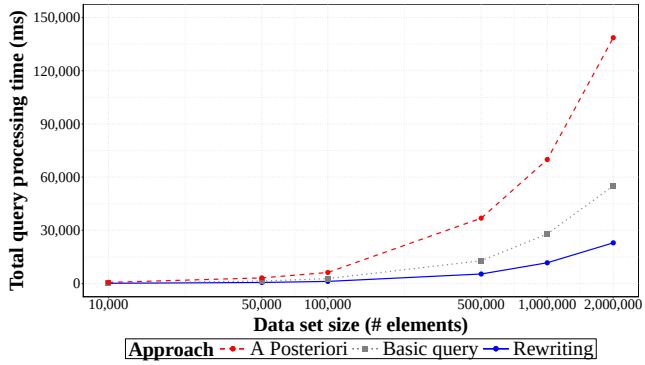


Figure 4.12 – The scalability of the rewriting approach compared to the a posteriori in-application processing and an unfiltered aggregation query on an Elasticsearch system. Note the logarithmic x-axis. Lower is better.

presumably due to the complexity of the aggregation operation. Moreover, the rewriting approach performs considerably better than the a posteriori filter.

For MariaDB, we notice a similar trend. However, the a posteriori filter processing times increased at a faster rate compared to the Elasticsearch test. This test indicates that the rewriting approach scales with regard to the data set size, and its performance is influenced by the data set, aggregation query, and database system.

For search queries, the scalability is expected to shift even more in favor of the rewriting approach, as the serialization of data items has a larger impact on the performance, especially for large data sets in which more data items need to be communicated over the network.

4.5.3 Pre-processing and a posteriori variations

Another part of the evaluation analyzes the impact of several variations of both the rewriting approach and the a posteriori filter on performance. In particular, for the a posteriori filter we regard the impact of policy reduction (as a result of substitution) on the performance, against policy evaluation over the non-reduced policy. For the rewriting approach, we considered the performance overhead of policy reduction as is also performed for the a posteriori filter, and query parameterization to increase security.

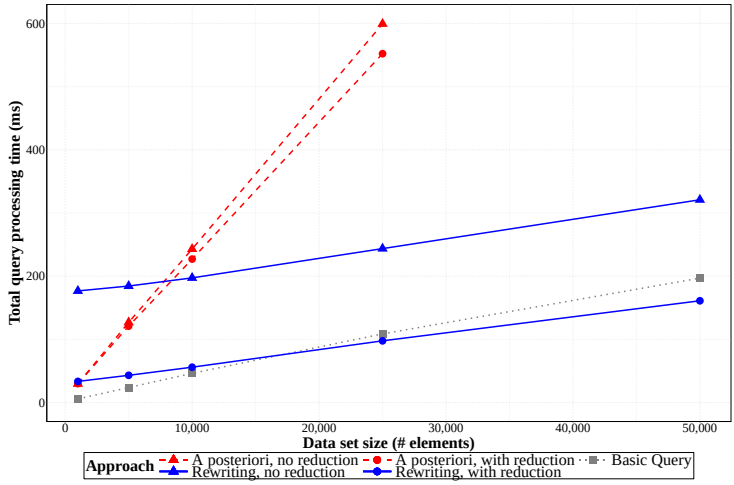


Figure 4.13 – Performance of several authorization enforcement variations for an increasing data set size on an MariaDB system. Lower is better.

Figure 4.13 compares the processing times of these variations for an increasing data set on a MariaDB system. The figure indicates that the rewriting approach performs better for reasonably sized data sets. Policy reduction for the a posteriori filter diminishes this issue to some extent compared to its non-reduction counterpart, but still increases faster than the rewriting approach variations. For larger data sets, the rewriting approach hence performs significantly better. Also, while policy reduction leads to lower processing times for this approach, the parameterization still yields reasonable results. Lastly, the basic query execution requires higher processing times than the rewriting approach with reduced policy from 25,000 data items onwards, as the result set must be communicated over the network, leading to higher communication overhead. Instead, the rewriting approach only communicates authorized data items, which leads to lower overhead.

4.5.4 Overhead analysis

For this part of the evaluation, we analyzed the overhead of several aspects for the rewriting approach and measured their impact on the total processing time. In particular, we considered the time required to perform partial substitution, the overhead to perform policy transformation and translation to a query, and the query execution and communication times.

Figure 4.14 provides an overview of the contribution of each step to the overhead of the rewriting approach for three different subjects. The subjects are entitled

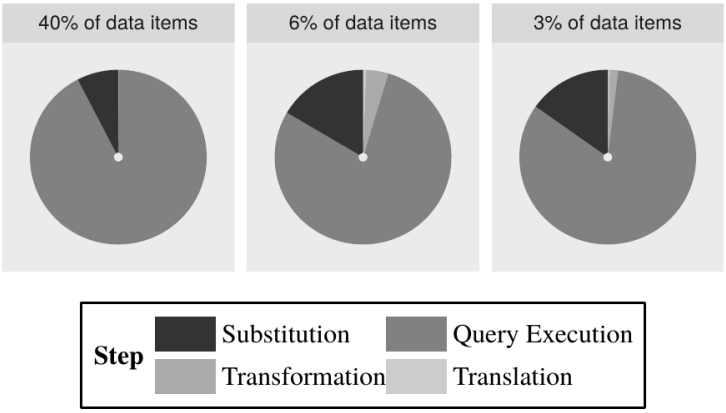


Figure 4.14 – The overhead involved in the approaches with partial substitution. Transformation and substitution overheads remain fairly constant for increasing data sets, whereas other factors are heavily influenced by the data set size.

to 40%, 6%, and 3% of a data set of 25,000 items, respectively. The figure demonstrates that query execution, which includes the processing of the query, serialization of the data, and network communication dominates the total processing time. This step causes about 79% up to 92% of the overhead. The partial substitution step, which minimizes the policy, is the second biggest contributor to the overhead, imposing about 8% to up to 17% of the processing time. Transformation and translation steps both had negligible impact on the total overhead.

These figures indicate that the subject attributes have a significant influence over the total processing time. This is due to the impact they have on the generated queries. The subject attributes can drive the complexity of queries, which in turn may increase the impact of query execution on the total processing time. For example, the subject that was entitled to 6% of the items has a proportionally lower query execution impact than the 3% subject, whereas the total processing time for the latter is in fact lower.

For increasing data sets, the proportional impact of the query execution on the total processing time also increases. This may be evident, as both transformation, translation, and partial substitution overheads remain the same no matter the size of the data set, and the actual execution time of the generated query will have an increasingly large cost. Also, the partial substitution may be amortized among multiple similar requests of the same subject, further reducing the impact of that step and optimizing the rewriting approach altogether.

4.5.5 Attribute impact

In order to expose the impact of policy properties on the processing time of the rewriting approach and a posteriori filter, this test assessed the total processing times involved for both approaches with an artificial policy that has an increasing number of different attributes which are all involved in the decision process. For this test, we generated several policies that require n different resource attributes to be evaluated to come to a decision, with n ranging from 5 up to 50 attributes. The tests were performed on a data set of 10,000 items, with a 50% proportion of entitlement.

Figure 4.15 shows the result for this test for an Elasticsearch database. Both MariaDB and Elasticsearch results indicated a similar trend. The rewriting approach performs better than the a posteriori filter. While the difference in processing times remains fairly constant, the slight increase suggests that the impact of the amount of different attributes remains limited. Moreover, our case studies suggest that the amount of attributes used in a policy typically remain limited (e.g., the eDocs policy considers 26 attributes).

Because all data items, including their attributes, need to be retrieved from the database, communicated over the network, and evaluated against an authorization policy, the a posteriori filter involves a higher overhead. For the rewriting approach, the curve is explained due to the time required to perform the transformation. Without network communication overhead, the transformation time even amounted to about 78% of the total processing time.

4.5.6 Policy complexity

This part of the evaluation analyzed the impact of the policy complexity on the total processing time for the three supported combining algorithms. In particular, we evaluate this for the `deny overrides`, `permit overrides` and `first applicable` combining algorithms and compare the rewriting approach with the performance of the a posteriori filter.

The test generated policies for varying policy tree *depths*, i.e., numbers of nodes on the path from the root policy component to the leaf-level components. For each depth, the policy contains $2^{d+1} - 1$ policy elements, 2^d of which are rules. Each component has two children and the same combining algorithm. Leaf components all have one permit and one deny rule. The test was performed on a data set of 1000 data items.

Figure 4.16 shows the processing time for the rewriting approach for different

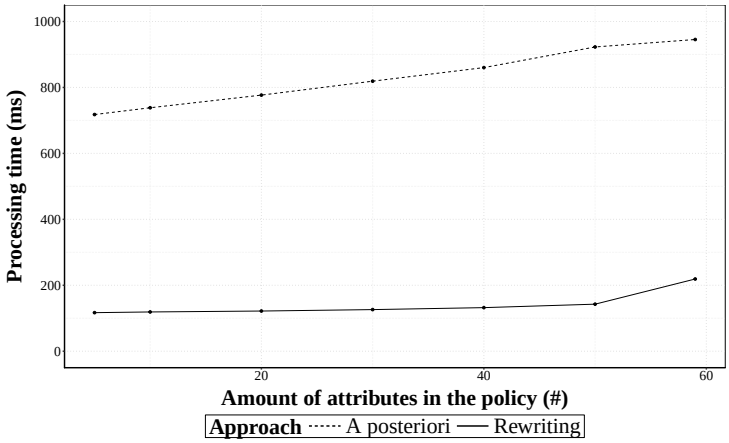


Figure 4.15 – The Elasticsearch processing times in function of the number of different attributes in a policy. All attributes are involved in the policy evaluation. Lower is better.

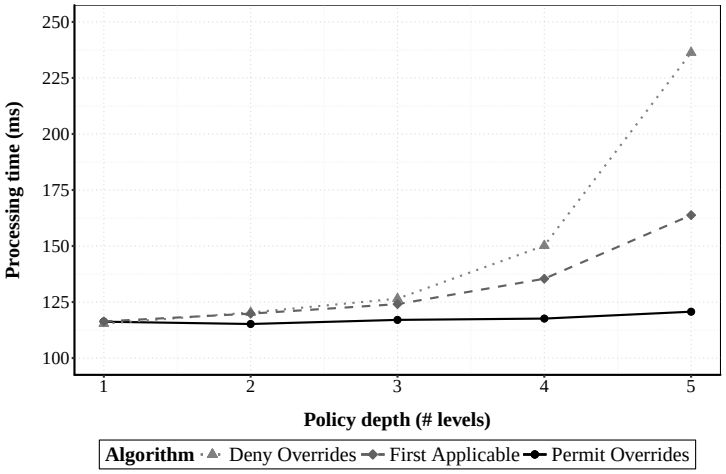


Figure 4.16 – Processing times on MariaDB for queries that were generated from balanced policy trees with an increasing amount of policy elements. Lower is better.

combining algorithms when the depth of the policy tree increases. The rewriting approach performs worst for the `deny overrides` algorithm. This is because the transformation introduces inevitable redundancy in the query to maintain original semantics for this combining algorithm[4]. This test, however, involves the worst-case scenario for a large policy. Moreover, our tests indicate that the a posteriori filter still exceeds the query rewriting approach for the most extensive policy of this evaluation. The largest proportion of the overhead is due to policy transformation, which amounts to about 85% of the total time for `deny overrides` components in the rewriting approach.

The `permit overrides` transformations, on the other hand, perform best because the transformation does not need to take into account the conditions of any deny rules.

4.5.7 Proportion of entitlement

This test measured the influence of the proportion of data items of the data set to which the subject has privileges (also referred to as the *proportion of entitlement*) on the processing time. This test used the same set-up as the attribute impact evaluation, with a data set of 10,000 items and an artificial policy that considers 50 different attributes.

[4]For more details, we refer to [38].

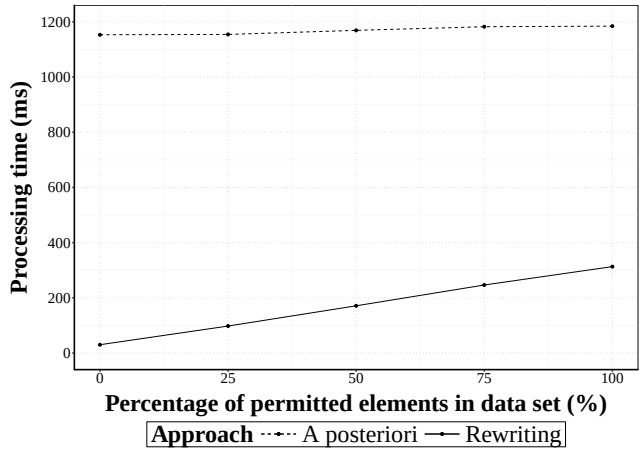


Figure 4.17 – Proportion of entitlement impact for a 10,000 item Elasticsearch data set and an artificial policy. Lower is better. For an increasing proportion of data items to which a subject has privileges, the processing times increase as well.

Figure 4.17 shows the processing times for an increasing proportion of entitlement for an Elasticsearch database. The tests performed for MariaDB indicated a similar trend. Processing times for the a posteriori filter remain fairly constant, as all data items that satisfy the search constraints require policy evaluation to determine whether the subject is privileged to access them.

However, for the rewriting approach, we notice that the processing time is directly proportional to the amount of data items to which the privileges exist. In other words, for a 25% proportion of entitlement, the processing time for the rewriting approach is lower than for 75%. This is caused by the lower network communication overhead due to data items that the subject is not privileged for. Even if the subject is privileged to all data items, however, the rewriting approach performs significantly better than the a posteriori filter.

4.6 Discussion

Performance. The evaluation results indicate that generally, the rewriting approach adopted by Sequoia yields a lower enforcement overhead than the a posteriori filter. While native database system authorization may still result in less overhead, these technologies do not support multi-organizational applications. The tests in the evaluation are susceptible to configuration adjustments and subtle optimizations. For example, database caching was

disabled as much as possible, though in real-world cases this may actually improve results for SEQUOIA compared to the a posteriori filter, as similar queries will yield lower processing times. Moreover, no query planning optimizations were performed, and it would be interesting to study its impact on performance of the rewriting approach. Such optimizations, however, are generally highly database-specific and were thus outside the scope of the evaluation.

The evaluation did not analyze the impact of simultaneous requests on the performance of SEQUOIA. Because the authorization rules are included in the rewritten query, the complexity of the query is increased which impacts performance. While partial substitution may reduce the eventual query to a considerable extent, this reduction is highly dependent on the attributes of the acting subject and the performed action. As a consequence, the rewriting approach may introduce significant overhead with regard to the scalability of requests when complex queries are performed. Performance monitoring and query analysis techniques may alleviate this issue and optimize this process. This requires an extension of the SEQUOIA architecture with a component that continuously monitors the performance of the underlying database and the queries that are submitted to that database. The extent to which such a component may alleviate this issue is an important part of future research.

The results demonstrate that even in absence of further optimizations, the rewriting approach scales better with regard to the data set size than the a posteriori filter. This is true even when the a posteriori filter is parallelized, due to the serialization and communication overhead in multi-tier architectures.

Security trade-offs. The extensive study on performance also analyzed the trade-off between performance and security. Like AMUSA enabled security experts to make this trade-off on a per-attribute basis, they are supported in doing the same for SEQUOIA with regard to queries.

In particular, the policy minimization process can significantly reduce the queries that are generated. In the eDocs case studies, this leads to reductions of up to 92% of the original policy. In larger multi-organizational policies, this is likely to have a significant impact if many organizations are involved.

However, policy reduction also introduces security risks if the original search parameters or the subject attributes cannot be fully trusted. Because the generated query is evaluation-context specific, they cannot be parameterized and cannot support the same security guarantees as prepared statements. Policy parameterization, on the other hand, yields a query that is reusable across different evaluation contexts, but can result in significantly larger queries.

A middle ground between these approaches, in which trusted and validated parameters and attributes drive policy minimization, and remaining search

parameters and attributes are parameterized, may alleviate this issue.

Limitations. While the rewriting approach does indicate promising results, there are also limitations and challenges for SEQUOIA.

First, obligations are not supported, as such specification capabilities are generally not provided by the query languages. This does not necessarily constitute a problem, however, as the case studies did not suggest a need for obligation support.

Second, our contribution focuses on supporting three prominent combining algorithms: `permit overrides`, `deny overrides`, and `first applicable`. These combining algorithms require specific transformation functions to convert the policy components to a boolean expression. Hence, other combining algorithms and alternative conflict resolution approaches require specialized transformation functions, and may not even be fully transformable. For example, the semantics of `only one applicable` combining algorithm supported by XACML yields an error if more than one child element is applicable. This is not expressible as part of boolean logic. SEQUOIA requires measures to deal with such situations and clearly define its behavior in these cases.

Third, the rewriting approach is also limited with regard to support for several database models. Some database models, particularly wide-column stores (e.g., Cassandra [224]) and key-value stores (e.g., Redis [188]), typically enhance scalability through constraining expressiveness capabilities of their queries. As a result, boolean expressions are not supported in full in their query languages. This establishes a hard limitation to the extent to which the rewriting approach can be applied in those systems. While use cases of key-value stores are typically not focused on multi-organizational applications, this does impose limitations to wide-column stores. Future research should investigate how the rewriting approach can be partially applied and complemented with an a posteriori filter to alleviate this issue.

Fourth, resource attributes that are mapped onto actual fields in the database could also lead to errors. For relational databases, this could be detected quickly and even automatically as it involves a fixed data model to which all data items must comply. However, for semi-structured data, such as is present in document stores, this may cause issues as there may be data items for which a mapped to field is not present. SEQUOIA handles this problem by excluding such data items from the result set, but approaches that support feedback to security experts may be required to prevent unexpected behavior.

Comprehensive enforcement. Overall, through its support for comprehensive enforcement of authorization technologies such as policy-based access control, attribute-based access control, and policy trees, SEQUOIA effectively

manages to support multi-organizational requirements such as isolation, sharing, run-time customization, and separation of concerns for data-focused operations while not imposing a prohibitive performance overhead. As such, it provides a promising contribution to access control in multi-organizational applications.

4.7 Conclusion

This chapter presented SEQUOIA, a data access middleware for authorization enforcement on data-focused operations in database systems. SEQUOIA uses technologies such as policy-based access control, attribute-based access control, and policy trees as a basis to restrict access to data items stored in underlying database systems.

Using a query rewriting approach, SEQUOIA performs run-time injection of the appropriate authorization rules into the original search and data aggregation queries based on dynamic run-time conditions. As such, the middleware leverages the filtering capabilities of the underlying database system to enforce authorization by selecting only data items to which the acting subject has the proper privileges.

Our evaluation shows that the approach introduces a low processing overhead and scales well with regard to the size of the data set on which the query is executed. Moreover, the approach performs even better in environments where the subject performing the query is only entitled to a small subset of the entire data set. The rewriting approach was also formally presented and verified in [38], and validated against several case studies of Chapter 2.

Through support for technologies such as policy-based access control, attribute-based access control, and policy trees in data-focused operations, the contributions of AMUSA are not limited to control-focused applications. SEQUOIA hence enables *comprehensive enforcement* in a transparent and extensive manner, thereby providing a significant contribution to authorization in multi-organizational applications.

Chapter 5

Entity-Based Access Control: expressive policy specification for multi-organizational applications

This chapter discusses Entity-Based Access Control (EBAC), a novel approach to specifying authorization constraints. EBAC considers entities as first-class citizens in the expressions that compose the authorization policy. By supporting both attributes and relationships, EBAC supports security administrators in reasoning more closely about the application domain, thus enabling more expressive authorization policies. As such, EBAC significantly improves the support for self-management of authorization in multi-organizational applications, by enhancing the expressiveness of authorization policies to align more closely with the organizational structure of the security administrator than supported by state-of-the-art technologies.

This chapter is based on a publication at the Annual Computer Security Applications Conference [34], and is structured as follows: Section 5.1 introduces the general problem and how it relates to the challenges in multi-organizational authorization. Section 5.2 presents a key scenario that is used as a running example throughout the chapter. Section 5.3 introduces a policy model that supports EBAC, discusses how entity-based expressions are evaluated, and presents a policy language that applies these concepts. Section 5.4 evaluates this policy language and compares it to XACML. Section 5.5 provides a general

discussion on the advantages and challenges of EBAC. Section 5.6 discusses related work. Section 5.7 concludes this chapter.

5.1 Introduction

Over the last decades, several access control models have been proposed that help reasoning about authorization constraints in a focused manner. Access control models provide a framework that aids in defining the link between privileges and subjects. On the other hand, policy models specify *how* such links are established, and thus form an essential aspect between the access control model and the mechanism that enforces the authorization specifications. Chapter 2 elaborated on the XACML policy model as fundamental building block for the previously introduced contributions in this thesis. Moreover, the chapter presented several prominent access control models. In particular, attribute-based access control (ABAC) and relationship-based access control (ReBAC) have been at the forefront of a lot of recent research [181, 209].

Essentially, ABAC reasons about privileges in terms of attributes, which can be regarded as key-value pairs assigned to subjects, resources, actions, and the environment that are compared to each other or to concrete values. Attributes are ubiquitous, and enable expression of fine-grained authorization specification that are rooted in the application domain. Also, because they are separated from the policy itself, attributes represent the characteristics of subjects, resources, actions, and the environment for which that policy specifies the conditions for authorization privileges. However, a limitation of ABAC is that it cannot reflect relationships in a straightforward manner. This is evident from authorization rule 5.1, which is based on the eHealth case of Chapter 2.

> *"A physician can read medical records that correspond to consultations for which the treating physician is of the same hospital, and corresponding to a patient that is also enrolled to that hospital."* (5.1)

This rule implicitly evaluates the relationship of physicians with a patient's health record, and its translation to ABAC involves attributes such as `resource.consultation_physician_affiliation` and `resource.consultation_patient_enrollments`. Such synthetic attributes implicitly reflect the relationship that the resource has with concepts in the application domain. Obtaining values for these attributes from the underlying data model requires custom specification of complex queries by a security administrator, and may hence violate the principle of separation of concerns. This is impaired further

for rules that force such attribute queries to implicitly reflect the access control logic, as is the case for rule 5.2.

"A physician can read a medical record of a patient if that physician had another consultation with that patient in the last year." (5.2)

In rule 5.2, the consultations in the last year can be represented by an attribute `subject.consultation_patients_last_year`. However, this attribute already reflects the authorization constraint rule 5.2 aims to enforce. This introduces several issues. First, it potentially reduces the readability of the authorization policy. Attribute names can sometimes insufficiently reflect the intent of the constraint, yielding false expectations. Second, whenever a rule changes, both the authorization policy as well as the attribute query need to be adjusted. For example, consider we want to modify rule 5.2 to permit access two years after the consultation. In this case, both the attribute name `subject.consultation_patients_last_year` as well as the underlying query should be adjusted to reflect this. This violates the principle of separation of concerns, because rule alterations may involve not only the security administrator, but also a database expert. Third, since changing a rule may require modifying the query that retrieves its corresponding values, this may even necessitate recompilation and redeployment of the application. This is a problem especially for applications that require *near-continuous up-time*, which is a common requirement in multi-organizational applications.

One way to address these issues is through adopting a relationship-based access control (ReBAC) model instead. As opposed to ABAC, ReBAC does support representing relationships in a natural manner. Essentially, ReBAC determines access privileges based on the existence of a relationship between the subject and the resource [70]. As such, ReBAC can support rules that can be expressed in terms of relationships straightforwardly. For example, rule 5.1 can be specified through simple relationship paths that indicate the treatments and enrollments of patients. However, while this facilitates expressing rules that can be specified in terms of relationships, ReBAC does not support natural comparison of values associated with the subjects and resources. For example, rule 5.2 involves temporal restrictions, which can not straightforwardly be expressed in terms of relationships. Consequently, this complicates the task of security administrators in specifying policies that align closely to the structure of the organization they are responsible for.

In this chapter, we address these issues by proposing a novel authorization approach: Entity-Based Access Control (EBAC). EBAC considers entities as first-class citizens for access control policies. Entities can have both attributes and relationships, which are addressed as key-value pairs. Similar to ABAC,

EBAC compares attributes to each other or to concrete values in order to determine privileges. In contrast to ABAC, however, entity-based policies can also navigate relationships and compare attributes associated with related attributes along the path. This concept of relationships was inspired by ReBAC.

This chapter proposes a policy model that supports this concept. The policy model starts from the same concepts as XACML and STAPL with regard to structure and expressions, which were discussed in Chapter 2. However, comparisons of attributes associated with *entities* along the relationship path can be composed into boolean expressions that are the fundamental building blocks of the policy elements. Moreover, the policy model supports policy evaluation based on relationship paths of arbitrary length between entities and can reason about entities along such paths.

EBAC enables security administrators to specify policies that align closer to their organizational structure, as the expressiveness supports evaluation of properties associated with related entities. This enables rules such as rule 5.1 and rule 5.2 to be enforced by the authorization system, avoiding the need for *synthetic* attributes that need to be backed by custom queries. As such, EBAC increases expressiveness for security administrators, thus contributing to multi-organizational authorization, and improving self-management capabilities of organizations.

5.2 Key scenario

The contributions in this chapter apply to a wide range of scenarios, in particular in the domain of multi-organizational applications. For this chapter, however, we consider the eHealth case study that was introduced in Chapter 2.

The eHealth application manages medical records of patients for multiple health care facilities. The application, illustrated in Figure 5.1, enables physicians across health care facilities to look up and manage medical records for patients who participate in the system. These records are centrally stored and managed. Hence, authorization is enforced at this centralized application as well.

From an authorization perspective, medical records are the primary resources. Medical records describe the activities performed at a consultation and the results of medical treatments. A consultation may comprise multiple medical records. Patients can be enrolled to health care facilities to make this information available to all of their physicians. In this application, physicians are the primary subjects. They are affiliated with a health care facility and have multiple specializations. They can create and read medical records.

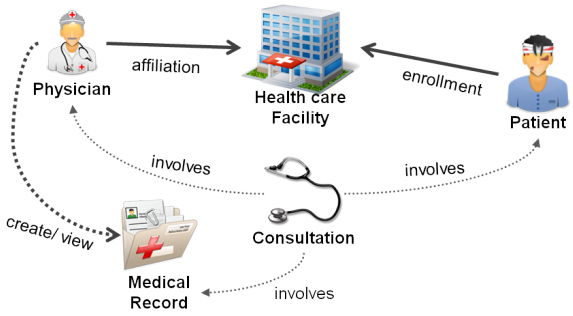


Figure 5.1 – The eHealth application has physicians as the primary subjects. For brevity, we do not consider authorization for individual patients in this chapter.

Evidently, authorization policies must restrict the actions of physicians in order to safeguard the patients' data. Health care facilities want to constrain access according to their requirements and internal structure. Figure 5.2 enumerates a non-exhaustive list of authorization rules that are illustrative to the application. While prominent models such as ABAC and ReBAC enable specification of expressive rules, not all rules described in Figure 5.2 can be specified according to these models.

In particular, an important drawback of ABAC is that it does not always relate seamlessly to the application domain about which it is supposed to reason. Because ABAC only focuses on attributes that are assigned to subjects, resources, actions, or the environment, auxiliary data entities are not considered in the authorization rules. Additionally, relationships can not be expressed straightforwardly in terms of attributes. For example, consider `resource.consultation_physician_affiliation_id` an attribute that reflects the hospital of a physician who created a medical record. While such an attribute identifies the intended hospital, it also requires that a custom data query is specified that retrieves its corresponding values. Hence, this involves a security administrator that manually maps the attributes onto the data model.

Translation of complex relationships also introduces this mapping issue. For example, consider rule 9 from Figure 5.2. A direct supervisor of a subject can be modeled as an attribute `subject.supervisor_id`. However, such an approach cannot be taken for *indirect* supervisors, which instead requires workarounds such as an attribute `subject.supervisor_ids` that represents all (in)direct supervisors of the subject. In order to do so, security experts need to map the recursive relationship between physicians over a supervisor relationship onto the data model.

1. A trainee physician can not create medical records.
2. A physician can read a medical record if the patient has granted his/her explicit consent to do so.
3. A physician can read a medical record if he/she is supervisor of the physician that created the record.
4. A physician can read a medical record if the corresponding patient had a consultation with him/her in the last year.
5. A physician can create a medical record if the categories he/she assigns to the record are all specializations of the physician.
6. A physician can create the medical record of a patient if that patient is enrolled to the same facility with which that physician is affiliated.
7. A physician can read a medical record if it corresponds to consultations for which the involved physician is from the same facility, and corresponds to a patient that is also enrolled to the same facility.
8. A trainee physician cannot read a medical record if it corresponds to consultations that took place more than four years before the trainee started.
9. A physician can read a medical record if the patient of its corresponding consultation already had a consultation with the physician's (in)direct supervisor.

Figure 5.2 – List of key authorization rules for a hospital that uses the eHealth application to manage medical records of patients.

Moreover, rules that reason about multiple attributes of a single entity that are addressed over a relationship are not supported by ABAC. For example, consider rule 4 from Figure 5.2. This rule could be described using an attribute such as `subject.record_ids_of_last_year`, that holds all consultation identifiers for which the subject was involved in the last year. However, this also requires the specification of a custom data query to retrieve the values. Worse, such query will at least partially reason about the rule for which the attribute is being used. This creates problems when this rule must be adjusted to permit access to medical records corresponding to consultations in the last *two* years, as this would involve a modification of both the query as well as the attribute that is referred to in the policy. Such an adjustment may require recompilation and redeployment of the application, which violates the near-continuous up-time requirements that are typically present in multi-organizational applications.

While ReBAC may partially address these issues through embracing relationships, their lack of support for attribute comparisons and temporal constraints, which are required for rules 4, 5, and 8 of Figure 5.2, create their own drawbacks.

Consequently, existing models lack the expressiveness to specify these rules. However, in order to fully support multi-organizational requirements such as self-management, expressiveness of authorization policies should be maximized.

5.3 Policy model and language

In order to address the issues with existing access control models, we present Entity-Based Access Control (EBAC). EBAC generalizes ABAC and ReBAC by adopting an entity-relationship model as a basis for the policy expressions. These expressions reason about relationships between entities, and support comparing the attributes of the entities along the relationships to determine privileges. EBAC integrates two core concepts:

ER model. The entity-relationship model (ER-model, [58]) is a data model that supports the representation of application domains. The ER-model comprises three components that are combined into the model: (i) entities, (ii) their properties, and (iii) relationships between the entities. Whereas the properties assigned to entities are similar to the attributes of ABAC, the relationships match those of ReBAC.

Logical expressions. Like the XACML policy model, boolean expressions compose rule conditions and policy component targets in the EBAC policy model. These expressions compare attributes associated with entities with each other and with concrete values. Boolean expressions can also be compounded through logical connectives (e.g., and, or, not). Moreover, EBAC supports predicate logic functions $\forall$ and $\exists$ which reason about relationships and attributes of the entities among these relationships.

This section first outlines how these concepts are integrated in the policy model for EBAC. Next, it describes a language that is based on this policy model.

5.3.1 Entity model and instantiation

To specify policy expressions, an *entity model* must exist that describes what can be compared in these expressions. An entity model defines all types of entities, their attributes, and relationship types that can exist between any two entity types in the context of a certain application domain. Entity models correspond to the ER-model, which supports entity sets (i.e., *types*), their corresponding properties and relations to be specified [58]. They were inspired by the ReBAC model of Crampton and Sellwood [70]. However, unlike Crampton and Sellwood,

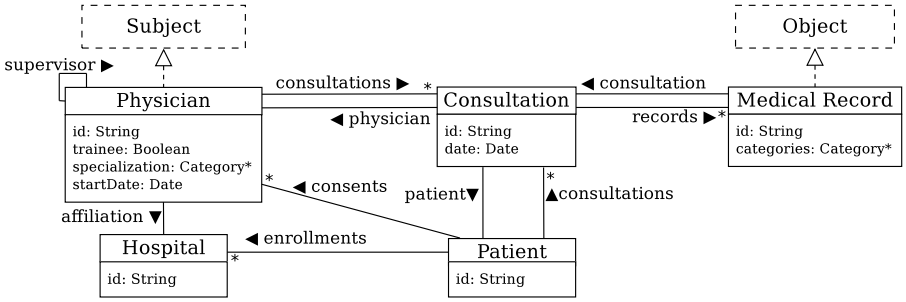


Figure 5.3 – Example of an entity model for the key scenario. For brevity, this figure omits action and environment entities. Also, the arities for relationships are not shown explicitly. Instead, a $*$ indicates a minimum arity of 0 and unbounded maximum. Otherwise, the relationship type occurs exactly once.

our approach also supports attributes for the entity types, and restricts arity and type of both attributes and relationships. This enables policies to align closely to the underlying application domain because the entity model takes into account any number of relationships and attributes in any form for each relevant entity type.

Entity models restrict policy expressions, which are specified in the context of an application domain. In particular, an entity model defines the entity types, their attributes, and relationship types that can be specified in policy expressions. Moreover, it specifies data types so that restrictions can be imposed on comparisons between any two attributes. In fact, such an entity model composes a *policy domain*, which is the subset of the application domain that is relevant for reasoning about application-level authorization.

Entity graph

Consider $G(V, E)$ a directed, multi-labeled graph that represents the entity and relationship types for a certain policy domain. In $G(V, E)$, vertices represent entity types and the edges represent relationship types. Figure 5.3 illustrates an example entity model that is based on the key scenario, and which can be represented in a directed, multi-labeled graph. Similar to Crampton and Sellwood [70], edges of G are labeled. These labels encompass the set R. Relationship labels describe the type of the relationship between two types of entities and includes the name and arity for that relationship type. Contrary to Crampton and Sellwood, EBAC also supports the assignment of labels to vertices in G in order to reflect their attributes. Each vertex can be

assigned multiple such labels. Consider L the set of all attribute labels assigned to the vertices $v \in V$. Each label of L is a tuple (v, k, t), for which

- Element $v \in V$ represents the vertex to which the label is assigned.
- Element k references the attribute and is unique for the vertex v.
- Element $t \in T$ identifies the data type of the attribute. We define T the set of all supported data types in the entity model.

An entity model $S = (G(V, E), L, T, R)$ reflects all entities that can be evaluated in the expressions of a policy, together with their attributes and relationships. For example, for the entity model represented in Figure 5.3, the set of attribute labels L includes `startDate` for its consultations, the set of supported data types T includes a `Category` data type which has a finite set of typed values, and the set of relationship labels R includes restrictions such as that the `affiliation` relationship is present exactly once for each physician. As such, an entity type is defined by its vertex v, the set of outgoing edges, their accompanying relationship labels from R, and the attribute labels from L.

Instantiation

Policy evaluation requires that attribute values are retrieved for the relevant entities. The existing entities of an application form the *instantiation* of an entity model. Figure 5.4 illustrates an example of an entity model instantiation based on the entity model of Figure 5.3. An instantiation $G_I(V_I, E_I)$ reflects the collection of all instances of each entity type and their intermediate relationships

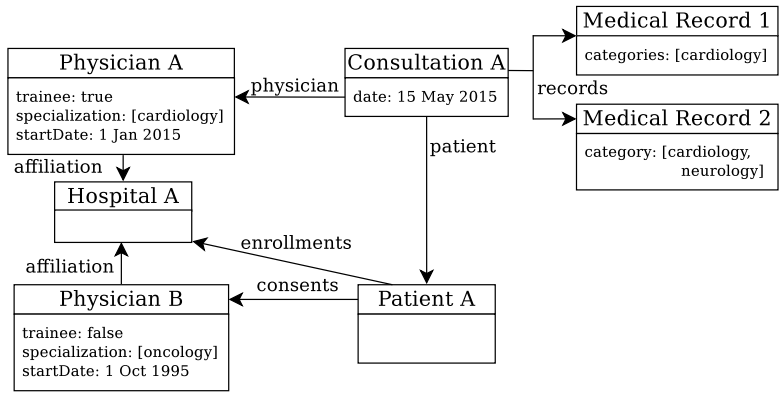


Figure 5.4 – Instantiation of the entity model from Figure 5.3. Physicians A and B are both affiliated with the same hospital, but hold different (in)direct relationships with Patient A.

that are present in an application. This includes the instantiation of entity type attributes (v, k, t) as tuples (i_v, k, val) in which:

- An entity $i_v \in V_I$ that corresponds to an entity type bound to a vertex $v \in V$ from the entity model.
- A key k that corresponds to the attribute in the model.
- A concrete value val of type $t \in T$ which corresponds to the attribute for this instance.

The model instantiation represents the context from which policy evaluation is performed. While such policies are specified in terms of the entity model, the evaluation deals with a concrete instantiation that takes into account the entities related to the relevant subject, resource, action and environment over the pre-defined model. As such, it extends the attribute-based policy evaluation context with a set of concrete entities and their corresponding attribute values, whereas attribute-based evaluation only considers attribute values associated with the subject, resource, action, and environment in its context.

5.3.2 Expressions

The fundamental building block of the policy model is the boolean expression. Our policy model supports three types of expressions: (i) simple comparison expressions, which support comparison between two attributes or between an attribute and a concrete value; (ii) composed expressions, which compound other expressions by means of logical connectives (i.e., and, or, not); and (iii) quantifier expressions, which introduce quantifier functions that reason about properties and relationship paths of arbitrary length. Attributes are addressed in these expressions through *path selectors*.

Path selectors

Access requests always involve a *subject*, *resource*, and *action*. Moreover, a request *environment* may be included to reflect the context in which the request is performed. In order to retrieve attribute values, they must be addressed starting from one of these four *base entities*. However, since EBAC supports auxiliary entities that have their own attributes associated with them, a mechanism is required to address these entities as well. Through path selectors, relationships can be traversed in order to assess attributes that are associated with the related entities.

A path selector refers to an attribute through a chain of relationship types that starts with one of the base entities and ends with an attribute of the

addressed entity. For example, `resource.consultation.patient.id` refers to the identifier that corresponds to the `patient` of the `consultation` to which the base resource relates.

A *path selector* is a tuple (s, P, k) in which

- The entity instance $s \in S$ is the *starting node* from which the path is traversed, with S a finite set of instances of base entities of a type in $\{subject, resource, action, environment\}$.
- A finite, ordered list $P = (p_1, p_2, ..., p_n)$ contains the *path* of relationship types (each occurring exactly once) that must be traversed in order to be able to refer to the attribute. The order of P must comply to the entity model.
- The key of the attribute k that is assessed from the addressed instance at the end of the path P. An attribute by this key must be defined at this end node.

For example, a path selector that is informally specified as `resource.consultation.patient.id` is represented by $(medrec_1, (consultation, patient), id)$ for a medical record $medrec_1$. If during evaluation the path could not be traversed because no instance of the relationship exists, the expression that incorporates the path selector evaluates to `false`.

Path selectors enable traversal of relationship paths in order to assess an attribute that can be compared as part of an expression. Although the remainder of this chapter regards attribute comparisons, both references to attributes and the policy evaluation process will always consider path selectors. As such, they form a key concept in the policy model for EBAC.

Simple Expressions

Simple expressions are the basic components of any composed expression. They compare two attributes or an attribute and a literal value. For example, consider expression 5.3:

$$\text{`}neurology\text{'}\ in\ resource.consultation.physician.specialization \tag{5.3}$$

Expression 5.3 compares a concrete value (i.e., '*neurology*') to a set of specialization values and its evaluation returns `true` when the evaluation context contains a resource that holds a relationship path to a physician which has this specialization.

Simple expressions are tuples $(l, r, \diamond)$ for which:

- Elements l and r indicate the left and right leg of the comparison, respectively. The left leg is always an attribute, whereas the right leg can be either a literal value or an attribute. The attributes for l and r are specified by means of path selectors.
- Comparison operator $\diamond$ (e.g., $\leqslant$, $=$) indicates the comparison operation that is to be performed between the elements.

Both l and r should be of the same type $t \in T$, or, alternatively, $\diamond$ is an operator that can handle a comparison between two elements of different types (e.g., a list type supports the comparison operator in).

Expression 5.3 is modeled as tuple 5.4:

$$((resource, (consultation, physician), specialization), \text{`}neurology\text{'}, in) \quad (5.4)$$

In the example in tuple 5.4, the in comparison operator checks whether the category '*neurology*' is present in a list of categories that is retrieved by looking up the attribute for path selector `resource.consultation.physician.specialization`.

Composed Expressions

Simple expressions can be combined using logical connectives into composed expressions. For example, consider composed expression 5.5:

$$subject.trainee \ \wedge \ resource.consultation.date \ \leqslant \ env.now \quad (5.5)$$

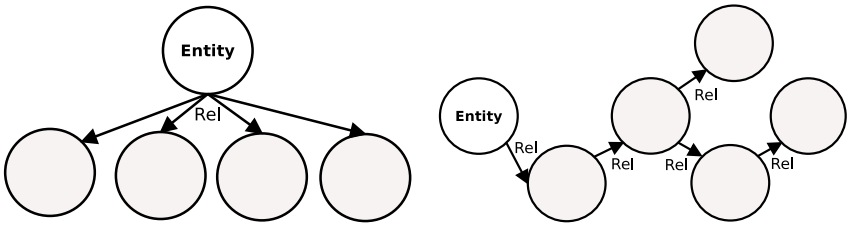
Expression 5.5 composes two simple expressions by means of a logical connective. A composed expression is either expression $\neg expr_1$ or a tuple $(expr_1, expr_2, conn)$ with

- Both $expr_1$ and $expr_2$ expressions.
- A logical connector $conn \in \{\wedge, \vee\}$.

Both $expr_1$ and $expr_2$ can be any type of expression, including other composed expressions. This supports composition of complex expressions.

Quantifier Expressions

In order to further leverage the concept of relationships, the policy model also enables reasoning about any entities that can be addressed through these



(a) Horizontal quantification **(b)** Vertical quantification

Figure 5.5 – Evaluation of relationship quantifier expressions. Horizontal expressions reason about relationships that occur more than once. Vertical expressions consider recursive relationships.

relationships. EBAC supports two such expressions: (i) *horizontal* and *vertical* quantifier expressions. These are illustrated in Figure 5.5.

Horizontal quantifier expressions. Entity types can specify relationships that occur more than once. For example, a patient can designate an explicit consent to multiple physicians to read his/her medical records. Similarly, entities may have attributes with multiple values. For example, physicians can be assigned multiple specializations.

EBAC supports quantifiers to reason about entities that are addressed in these relationships. For example, consider rule 5 of Figure 5.2. The condition expression for this rule is illustrated in expression 5.6:

$$\forall\ category \in resource.categories : category \in subject.specialization \quad (5.6)$$

Expression 5.6 reasons about a set of categories, and determines membership in `subject.specialization` for each value. If every category of the resource is also a specialization of the subject, the evaluation of this expression is satisfied.

Similarly, relationships with an arity higher than one enable reasoning about entities that are addressed by them as illustrated in Figure 5.5a. Consider rule 4 from Figure 5.2. This rule reasons about consultation properties and can be summarized in expression 5.7:

$$\exists\ consultation \in resource.consultation.patient.consultations :$$

$$\{consultation.physician.id == subject.id \wedge \tag{5.7}$$

$$consultation.date < (environment.currentDate - 1\ year)\}$$

Expression 5.7 traverses the relationship path `consultation.patient.consultations` for the resource and reasons about every entity that is addressed via this path. If for one of the consultations, the inner expression is satisfied (i.e., the subject is the treating physician and that consultation took place within the past year), the evaluation of this expression 5.7 returns `true`.

Quantifier expressions are enabled by using *partial expressions*. Partial expressions are referred to inside the quantifier ($\forall$ or $\exists$) and contain anonymous path selectors. While a regular path selector by which an attribute is selected includes a base starting node (i.e., a subject, resource, action, or environment entity type), an *anonymous* path selector supports *any instance* as starting node. Partial expressions are parameterized by combining the anonymous path selector with a path that starts from a base entity type. As a consequence, the policy evaluation can determine how to parameterize this expression by determining the relevant base type from its evaluation context.

For example, expression 5.7 includes the partial expression `consultation.physician.id == subject.id` $\wedge$ `consultation.date <  (environment.currentDate - 1 year)`. Because path selectors have a path starting at a subject, resource, action, or environment entity, the entity `consultation` is an *anonymous* path selector. Through parameterizing the anonymous path selector by every entity addressed by `resource.consultation.patient.consultations`, the attributes of expression 5.7 can be referenced. As a result, entity-based policies can reason about properties of entities in a relationship with an arity higher than one.

We define horizontal quantifier expressions as tuples $(prop, pexpr,\ f\ \in\ \{\forall, \exists\})$ in which

- Element *prop* is a sequence *Seq* of entities that are addressed via a relationship with an arity greater than one. Alternatively, *prop* can represent a multi-valued attribute instead, enabling concrete values to be compared to an attribute such as in expression 5.6.
- Element *pexpr* is a partial expression that is parameterized with each item $e \in Seq$. Anonymous path selectors must be of the same (entity) type than that of the elements $e \in Seq$.

- Function $f \in \{\forall, \exists\}$ indicates the quantifier type that is used for the expression.

Figure 5.5a illustrates how an entity may relate to other entities in horizontal quantifier expressions. This figure does not show the case in which concrete values are assessed. Horizontal quantification addresses a fundamental issue that is present in ABAC, because attributes associated with a single entity addressed over a relationship cannot be represented through multiple attributes. This is discussed further in Section 5.4.

Vertical quantifier expressions. Horizontal quantifier expressions reason about the relationship of an entity with a set of other entities. However, they do not support reasoning about entities among a recursive relationship. Consider again rule 9 from Figure 5.2. This rule cannot be translated in terms of a horizontal quantifier expression, because the `supervisor` relationship of a physician needs to be traversed recursively until (a) a supervisor physician is found for which the condition is satisfied, or (b) no supervisor exists along the relationship path, in which case the rule cannot be applicable.

To accommodate this, *vertical* quantifier expressions can recursively traverse a relationship, as illustrated in Figure 5.5b. Vertical quantifier expressions are evaluated for each entity on a recursive relationship path. In rule 9, this involves the physicians' recursive relationship `supervisor`. The inner expression is evaluated *for each physician* found while traversing this relationship, until the condition is satisfied or no more supervisor exists.

We define a vertical quantifier expression as a tuple $(es, P, pexpr, f \in \{\forall_\rho, \exists_\rho\})$ for which

- Element es specifies a starting *entity* selector. Similar to path selectors, entity selectors specify a path on which an entity is addressed. However, in contrast to path selectors, entity selectors do not address an attribute at the end of this path, but rather address the entity directly.
- The relationship path $P = (p_1, p_2, ..., p_n)$ a finite, non-empty, ordered list that specifies the path that must be traversed at each step in order to parameterize the partial expression $pexpr$. The last element p_n always ends at one or more entities of the same type as es.
- A partial expression $pexpr$ which is parameterized with each entity along the path.
- Function $f \in \{\forall_\rho, \exists_\rho\}$, with $\forall_\rho$ the equivalent of universal quantifier $\forall$ and $\exists_\rho$ the equivalent of $\exists$.

In order to bound evaluation, vertical quantifier expressions can optionally be extended by $min, max \in \mathbb{N}$, $min < max$ to indicate the minimal and maximal depth of the path for which the inner partial expression must hold.

Consider again rule 9, which concerns (in)direct supervisors of a physician. The condition for this rule contains a vertical quantifier expression illustrated in expression 5.8.

$$\exists_{\rho,supervisor}\ sv\ \in\ subject.supervisor : \exists\ cons\ \in\ sv.consultations : \\ (cons.patient.id == resource.consultation.patient.id) \tag{5.8}$$

During evaluation, this policy engine traverses the subject's supervisor, evaluates the partial expression (starting from the horizontal quantifier expression that comprises the inner expression for the vertical quantification). If no match is found, the same partial expression evaluation process is repeated for `subject.supervisor.supervisor`, and so on. This evaluation process is repeated this until (a) no more supervisor exists, (b) the inner partial expression is satisfied, or (c) a previously encountered physician is found (which indicates a loop in the recursive relationship).

While the `subject.supervisor` relationship has an arity of 1, relationships with a higher arity could also be supported for vertical quantification. In this case, each entity from that relationship is used as parameter for the inner partial expression. If no entity is found that leads to a definite answer for the evaluation, each of the entity's children are evaluated in a breadth-first manner. This process is repeated for each element until no new entities can be found. Similar to a singular recursive relationship, duplicate entities are not revisited.

Vertical quantification expands the expressiveness of a policy by leveraging recursive relationships. For ABAC, this cannot be addressed in the policy itself, but rather requires attributes that reflect the recursive relationship in their entirety. This is discussed further in Section 5.4.

5.3.3 Policy language

In order to support actual policy specification and enforcement, the concepts introduced in the previous sections must be composed into an entity-based policy language. This section introduces such a language, which was based on STAPL [79]. This language, called *Auctoritas*, accommodates the concepts of expressions and an evaluation model that is based on the entity model and its instantiation as described previously. As such, we provide an authorization system and policy language which leverages the concepts of the original STAPL policy model (similar to XACML), but is extended to support entities and relationships.

```
Rule("rule_1") := deny iff (subject.trainee)
Rule("rule_2") := permit iff (subject.id in resource.consultation.
    patient.consent)
Rule("rule_3") := permit iff (subject.id === resource.consultation
    .physician.supervisor.id)
Rule("rule_4") := permit iff (subject.consultations.exists(
    consultation => consultation.patient.id === resource.
    consultation.patient.id & environment.currentDate < (
    consultation.date + 1.years)))
Rule("rule_5") := permit iff (resource.consultation.categories.
    forall(cat => cat in subject.specializations))
Rule("rule_6") := permit iff (subject.affiliation.id in resource.
    consultation.patient.enrollments)
Rule("rule_7") := permit iff (resource.consultation.physician.
    affiliation.id === subject.affiliation.id & (subject.
    affiliation.id in resource.consultation.patient.enrollments))
Rule("rule_8") := deny iff (subject.trainee &
    subject.startDate > (resource.consultation.date + 4.years))
Rule("rule_9") := permit iff (subject.supervisor.existsOnPath(
    supervisor => resource.consultation.patient.consultations.
    exists(cons => cons.physician.id === supervisor.id)))
```

Figure 5.6 – Translation of all rules of Figure 5.2 in the Auctoritas policy language. For brevity, we have omitted the prerequisites for the resource type to be a medical record, and the actions.

Figure 5.6 demonstrates the rules from Figure 5.2 translated in the Auctoritas policy language. Each rule can specify a condition and effect. For example, rule 1 in Figure 5.6 specifies that policy evaluation must yield a `deny` decision whenever the subject is a trainee physician (i.e., the value of the *trainee* attribute is `true`). If this is not the case, the rule is `not applicable` for that policy evaluation. This is similar to the STAPL policy language.

Auctoritas also supports path selectors to traverse an arbitrary number of relationships. Consider rule 3 in Figure 5.6. Starting from a resource, this rule traverses the `consultation`, `physician` and `supervisor` relationships, respectively, to access the `id` attribute. If any of these relationships is not present, the expression evaluates to `false`. The traversed relationship types in this example all have a maximum arity of 1. If a path selector contains a relationship type with a higher maximum arity, more elaborate logical functions are required.

Higher arity relationships are supported by quantifier expressions. For example, rule 4 in Figure 5.6 is supported by a horizontal quantifier expression that reasons about every consultation of the subject. The quantifier assesses each consultation and parameterizes the inner (partial) expression with it. The partial expression looks up the `patient` and `date` that correspond to the

consultation with which it was parameterized. This expression is specified as a lambda function that in practice assesses each identifier corresponding to the relationship it parameterizes. For example, rule 4 first looks up a list of all identifiers corresponding to the consultations of the subject, and for each consultation, assesses its attributes and relationships by providing the identifier and evaluating the expression. While rule 5 in Figure 5.6 also employs a quantifier, this involves only a list of values, as opposed to entities. As a result, the evaluation of rule 5 does not require such an elaborate evaluation approach. Rule 9 of Figure 5.6 also illustrates the specification of vertical quantifier expressions through the use of the `existsOnPath` keyword.

Policy specification requires the set-up of an entity model by a security administrator in order to utilize the concepts in the policy. The entity model maps the entities and their attributes to the underlying storage system in order to accommodate the retrieval during policy evaluation. For AMUSA, this could be automated for subject and organizational attributes, and resource attributes could be based on the application model. Our prototype implementation partially automates such retrieval through query composition that is based on the path selectors. While this supports fetching attributes in relational database systems, the actual specification and mapping of the entity model was not addressed as part of the prototype.

5.4 Evaluation

In the previous sections, we motivated the need for an increased expressiveness of access control policies. Through a key scenario, we described how, starting from an entity model and language extensions, the STAPL policy model can be adapted to accommodate an increased expressiveness. We also presented Auctoritas, a policy language based on STAPL that supports EBAC. In this section, we analyze the expressiveness of Auctoritas and evaluate the performance of a prototype authorization system that supports the language. We compare these results with XACML.

XACML is considered the de-facto standard policy language for attribute-based policies. For this evaluation, we regard this language as the representation of ABAC.

Table 5.1 – Comparison of expressiveness of XACML and Auctoritas with regard to the rules specified in Figure 5.2. A translation of these rules to Auctoritas was given in Figure 5.6.

Category	XACML	Auctoritas	Rules
Regular attribute comparisons	✓	✓	1
Composed relationships	×	✓	2, 3, 6, 7, 8
Simple horizontal quantification	✓	✓	5
Relationship horizontal quantification	×	✓	4
Vertical quantification	×	✓	9

5.4.1 Expressiveness

The expressiveness of a policy language determines the granularity at which authorization rules can be specified. In particular, we regard expressiveness of a language as the ability to specify authorization constraints without requiring any authorization logic to be adopted in other components (i.e., the authorization logic should be expressible in its entirety through the policy itself). As such, a need for *synthetic* attributes (which require adjustments in the query to retrieve its values) to express a certain rule indicates a gap in expressiveness.

We have analyzed the expressiveness for both Auctoritas and XACML. In order to compare them, we have translated the rules of Figure 5.2 into both of the languages. For Auctoritas, this was also depicted previously in Figure 5.6.

Table 5.1 provides a comparison of expressiveness of XACML and Auctoritas with regard to the rules that were specified in the illustrative example. The table indicates categories of expressions that can be specified in XACML and Auctoritas, respectively, with a ✓ if this does not require additional authorization logic outside the policy (e.g., by requiring the security administrator to provide custom queries or access logic for retrieving attribute values). Expression categories that cannot be expressed without additional authorization logic are presented with a ×. The table shows that XACML supports the first and third category, and as a result, can express rules 1 and 4 from the illustrative example. Auctoritas supports all categories of expressions. Hence, Auctoritas can express all rules from the illustrative example without requiring adjustment to the retrieval queries if the policy is changes. For this evaluation, we assume that the underlying data model for retrieving the attribute values coincides with that of the entity model that was presented in Figure 5.3, and that all attributes are stored in a database.

Table 5.1 demonstrates that for expressions that involve relationships, XACML falls short. Any rules that reason about relationships cannot be expressed

in XACML without additional authorization logic. Such rules could still be accommodated in this language through specification of custom attribute retrieval queries for any synthetic attributes that reflect these relationships. However, this decreases readability of the policy, and may require both the attribute and query to be reconfigured whenever a rule is modified. Such a rule modification may even require recompilation and redeployment of the authorization system, which affects the ability to support near-continuous up-time of an application.

Regular attribute comparisons. This category reflects the expressions that can be described using regular attribute comparisons. For example, the rule *"A trainee physician can not create medical records"* contains a condition that only addresses attributes of a subject, resource, and action directly. This expression category is supported by both XACML and Auctoritas, because policy changes that affect regular attribute comparisons can be applied without requiring change of the authorization system for both languages.

Composed relationships. XACML does not support the concept of relationships. As a consequence, composed relationships cannot be expressed in this language. Composed relationships regard comparisons that can traverse relationships to address attributes. For example, `resource.consultation.physician.trainee` traverses two relationships before addressing an attribute. In order to retrieve this attribute, an elaborate database look-up may be required. For XACML, this requires a custom query that reflects the traversal of these entities, whereas Auctoritas generates such a query out-of-the-box based on the mapping of the entity model.

Simple horizontal quantification. Similar to XACML, Auctoritas supports simple quantifications that reason about sets of values associated with a multi-valued attribute. This category involves horizontal quantifier expressions that only compare an attribute to all elements of a sequence, or compare all elements of two sequences with each other. For example, an expression may evaluate whether a value is greater than each element in a sequence. XACML provides functions such as `any-of`, `all-of` and `any-of-any` to accommodate such expressions.

Relationship horizontal quantification. This category involves expressions that assess multiple properties of a single entity that is addressed over a relationship with an arity greater than one. For example, the rule *"A physician can read medical records of patients who had a consultation with him/her in the last year"* assesses the `date` attribute and involved `patient` of the `consultations` relationship. In particular, (a) patients should have had a consultation with the physician; and (b) that same consultation should have occurred in the last year. These expressions must thus have a correlating

entity (i.e., the consultation), and therefore, cannot be evaluated separately. For XACML, this is an issue, as this correlating consultation cannot be modeled as a single attribute because two properties related to it must be evaluated. While it can still be expressed in XACML as part of a single attribute (e.g., `subject.patients_with_consultation_in_last_year`), any adjustments to the rule (e.g., consultations in the last *two* years), would require a change in both the attribute as well as in the custom query that supports its retrieval. In contrast, Auctoritas generates the queries to retrieve the corresponding attribute values automatically, and thus policy changes do not affect this.

Vertical quantification. This category involves expressions that reason about attributes of entities across recursive relationships. For example, the rule *"A physician can read a medical record if the patient of its corresponding consultation has previously had a consultation with the acting physician's (in)direct supervisor"* (rule 9) involves a recursive relationship `supervisor` which must be traversed until a physician is found that matches the treating physician of a consultation. In order to support this, XACML could model this rule through, among others, attribute `subject.indirect_supervisors`. However, this involves an attribute retrieval method that recursively obtains the supervisor of physicians until none is found. This may be problematic if any additional constraints are added to the (in)direct supervisors, for example whether they are still actively involved in the hospital. As a consequence, this also requires customized attribute retrieval queries in order to support translation in XACML, and is prone to adjustments not only in the policy but also in the authorization system. Auctoritas, on the other hand, can support the aforementioned additional constraints without any changes to the authorization system and is hence more expressive in this category as well.

5.4.2 Performance overhead

Policy evaluation that involves relationship traversal requires a more complex authorization system that supports a more elaborate authorization logic. This introduces policy evaluation overhead. In order to measure this overhead, we consider the performance of policy evaluation on the prototype authorization engine of Auctoritas and compare the results for equivalent rules in the XACML authorization system.

Our analysis is based on the translation of the rules from Figure 5.2. When the rules were not directly supported in XACML (as indicated in the previous section), we described the rules in terms of attributes that required custom queries that enabled the translation of the rule. For example, `resource.consultation.patient.id` is represented as an attribute `resource.consulta-`

`tion_patient_id` in XACML, which enforces the same authorization constraint but requires a custom attribute retrieval query. As argued earlier in this section, this may lead to problems with regard to adapting the rules. However, this part of the evaluation is concerned only with the performance overhead of the Auctoritas authorization system.

Set-up

For this part of the evaluation, we have considered the rules that were introduced in Figure 5.2 and presented for Auctoritas in Figure 5.6. Each rule was organized in a separate policy for which the evaluation times were measured for both XACML and Auctoritas. We did not regard the evaluation overhead of policies with multiple rules, nor did we take into account the parsing overhead of the policies (which is a known problem for XACML [227]). Rather, we focused on the overhead introduced by attribute retrieval and expression evaluation.

For attribute retrieval, JPA [178] was used to communicate with a MariaDB database containing the attributes for both languages, and all caching was disabled as much as possible. Attribute retrieval requests from the decision engine (PDP) to the retrieval components (PIPs) occurred using local TCP requests via Apache Thrift. For XACML, an optimized version of the SunXACML decision engine performed the policy evaluation.

The evaluation was performed on an Intel Core i5-3340M CPU @ 2.70GHz with 8GB RAM running Fedora 20. Each rule was evaluated for 100,000 applicable requests (i.e., the target matched, and as a result, attribute retrieval was required). The results omit the 2.5% minimal and maximal outliers of the measurements from the respective evaluation contexts.

Results

Figure 5.7 presents the results of the evaluation. This figure demonstrates that for most rules, Auctoritas performs significantly better than XACML.

For regular attribute comparison rules, the mean difference is 0.4ms in favor of Auctoritas. The simplified structure of Auctoritas as compared to the SunXACML implementation reduces its evaluation times.

In addition, composed relationships (i.e., rules 2, 3, 6, 7 and 8) evaluate significantly faster than the SunXACML implementation. This indicates that the number of relationships on a path does not have a great influence on the evaluation time. For instance, rule 7 has eight relationships that are traversed,

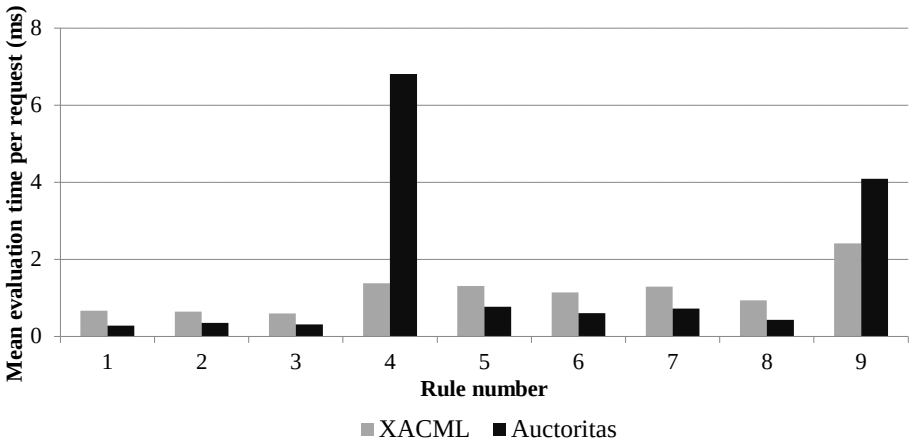


Figure 5.7 – Evaluation overhead comparison between XACML and Auctoritas. EBAC evaluation performs slightly better. Rules 4 and 9 introduce overhead due to the number of separate attribute look-ups required to evaluate the rules.

but does not involve significant overhead compared to rule 3, which requires only three relationship traversals. Auctoritas automatically generates queries to accommodate relationship traversals through joins that are based on the data model on which the entity model was mapped. This reduces the evaluation time (e.g., separately looking up each entity on a path for rule 3 resulted in a mean evaluation time of 1.58ms compared to 0.3ms).

However, Figure 5.7 also shows that XACML performs better for rules 4 and 9. These reflect the relationship horizontal quantification and vertical quantification expressions, and their evaluation introduces a larger overhead. For the horizontal quantification (i.e., rule 4), the overhead as compared to XACML is 5.4ms per request. In the prototype implementation, each parameterization of the expression involves a separate look-up for each entity. This introduces significant overhead compared to a query that filters out attribute values of entities that do not need to be evaluated, as in our XACML translation. The evaluated rule is inherently an expensive operation that may involve many consultation entities[1]. Similar to rule 4, vertical quantification (rule 9) introduced an overhead of 1.7ms compared to XACML. Evidently, the evaluation time will rise with larger data sets, and therefore such rules must be cautiously specified. Moreover, the Auctoritas authorization system did not apply any optimization techniques to reduce the overhead involved with these expressions. One approach to optimize the evaluation process for quantifiers would be to include context with the

[1]In this evaluation, the mean size of the operative data set was 23 consultations (i.e., patients had an average of 23 consultations associated with them).

attribute retrieval request, so that the authorization system can retrieve any entities that may be involved in the evaluation process in a single database request. However, regarding the complexity of these rules, we deem the overhead involved to be acceptable for the intended use cases of the language.

As a result, Auctoritas yields an acceptable overhead, performing even better than the SunXACML implementation for most rules of Figure 5.2. Moreover, since the evaluation Auctoritas is a prototype implementation, optimizations may yield improved results. As such, we consider EBAC to be a promising approach for specifying authorization policies for multi-organizational applications.

5.5 Discussion

Evaluation results. The results of the evaluation demonstrated that for relationship quantification, our prototype authorization system introduces a noticeable overhead compared to the SunXACML system. This was primarily due to the attribute retrieval operations being performed for each entity with which partial expressions were parameterized. This overhead can be addressed through optimization efforts, and hence we are convinced that EBAC provides a promising approach for authorization specification.

This conviction is also rooted in the observation that alternative approaches do not support the expressiveness that is provided by EBAC, as was also indicated in Section 5.4. As a consequence, in order to enforce the same authorization constraints, security administrators need to resort to custom query specification for attribute retrieval. This transfers part of the authorization logic of the policy to the PIP, reducing policy readability and complicating adaptability of the policy.

Especially for larger policies, this issue can be expected to increase as this would likely involve more elaborate authorization constraints and more attributes that need to be assessed. These attributes may in their turn implicitly reflect relationships with other entities, as we have noticed from our case studies. For the policy of the eDocs case study that was introduced in Chapter 2, for example, 10 out of 26 attributes implicitly reflected a relationship. Moreover, in the context of authorization middleware such as Amusa, in some cases it may not even be possible to alleviate the gap of expressiveness through custom attribute retrieval queries. This would impede self-management, because security administrators of respective organizations cannot be trusted to specify their own queries as this could pose considerable security risks, especially on a shared infrastructure.

In conclusion, while support for some expressions in EBAC may introduce an overhead, we are convinced this can be reduced through optimizations. Moreover, even without such optimizations we deem this overhead acceptable when the drawbacks of alternative approaches are regarded.

Challenges. While EBAC can thus enable more expressive policies, some challenges remain, especially in the context of multi-organizational authorization.

First, the increased overhead may affect the performance not only for the organization that specified the policy, but also for the other organizations with which an infrastructure is shared. This is a general concern for self-management, but is magnified by EBAC. For example, our evaluation demonstrated that for rule 4 of Figure 5.2, for a mean of 23 consultations, the overhead compared to XACML amounted to 5.4 ms. In some situations, horizontal relational quantification may involve more entities, leading to larger overhead. Hence, there is a need for performance isolation and optimization techniques to alleviate this issue, which may include attribute pre-fetching, attribute and decision caching, and evaluation parallelization. Moreover, approaches to shift evaluation of heavy-burden rules to after-the-fact audit can also reduce latency without removing accountability [53]. Alternatively, federated authorization approaches [78] may also entrust heavy-burden rule evaluation to the organizations that wish to enforce them in order to isolate performance issues.

Second, due to its flexible approach in addressing relationships between entities, EBAC is also susceptible to inference attacks. For example, consider the path selector `resource.creator.roles`. In a multi-organizational context that involves sharing, this could be used by organizations that have privileges to this resource to infer security configurations of the sharing organization. To prevent this, administrative policies could restrict attribute retrieval between organizations so that risks of inference are deterred. Such administrative policies could even apply the three-layered approach presented by Amusa. For example, by default, attribute retrieval could be limited to the organization that specified the policy, but exceptions could be specified by the provider and organizations for exposing a part of their own attributes. This may however increase the complexity of attribute retrieval. The impact of this and possible optimizations are interesting considerations for future work.

Third, increased expressiveness may also lead to inefficient policy specification. Due to the enabled traversal of relationships, elaborate entity models may provide several ways to specify the same authorization constraint, albeit with different impact on performance. For example, consider expressions `resource.consultation.patient.consultations.exists(c => c.physician.id === subject.id)` and `subject.consultations.exists(c => c.patient.id === resource.consultation.patient.id)` which both ap-

ply to the entity model of Figure 5.3. Both expressions are satisfied if a physician (i.e., the subject) has had a consultation with the patient associated with a medical record (i.e., the resource). However, evaluation of both expressions may vary depending on whether a physician has more consultations than the patient. Whereas such observations may be relatively straightforward in this scenario, finding the optimal specification may in some cases prove challenging. Through statistical analysis of entities and their relationships, an automated policy rewriting approach may optimize performance based on the underlying data model. Moreover, performance tactics used in graph databases [189] may also alleviate this issue further.

Lastly, the increased expressiveness also introduces additional constraints on *comprehensive enforcement* in database systems, as query languages of databases may not support the EBAC expressions. In particular, an open question remains what the extent is to which horizontal and vertical relationship quantifiers can be supported in query languages for prominent database models. For example, it may be impossible to transform an expression that uses vertical quantification over a recursive relationship to a SQL query, hence limiting comprehensive enforcement. This may require compromises in the enforcement approach that was discussed in Chapter 4.

Opportunities. While these issues constitute open research questions that will need to be addressed in future work, EBAC also provides significant opportunities.

First, the flexibility of EBAC can be leveraged to specify authorization policies that correspond more closely to the application domain. Historically, this has been an issue for many access control models. For instance, IBAC and RBAC only support explicit permission assignments between users and resources, which are not always necessarily grounded in any systematic assignment approach based on the properties and characteristics of entities in the application domain. While ABAC and ReBAC alleviated such issues to some extent through support for property-driven access control, they suffer from expressiveness gaps. This lack of expressiveness can impose compromises in policy specification, leading to policies that do not fully apply the authorization constraints to the characteristics of the application domain. For example, the relationship between business documents and organizations can not be leveraged in full for ABAC, requiring instead to use synthetic attributes to assess these organizations' properties. For EBAC, embracing the Entity-Relationship model [58] as a basis for the policy expressions ensures that application model concepts can be mapped intuitively on the policy.

Second, because EBAC enables properties and entities that correspond more closely to the application domain, (semi-)automated extraction of such properties

and entities from the application code is also possible. This enables application developers and security developers to set up an authorization model based on the application entity model without introducing prohibitive effort. For example, through annotations, a Java project could be instrumented in a way that the model can be extracted from the application code, automatically generating the queries required to retrieve properties that are used in the policy evaluation process.

Third, the expressiveness of EBAC is especially beneficial for multi-organizational authorization. For example, suppose that for the eDocs application of Chapter 2, a cable company wants to outsource the collection of outstanding payments of some customers to a debt collector organization. This requires a form of sharing that requires further restriction, e.g., sharing invoices of customers from a certain region (the cable company may not want to expose their entire client base) and to invoices that are past due date and yet to be paid. EBAC enables specification of policies that reason about such sharing in a more flexible and precise way. In particular, for this example, the regions of both customers and the debt collector are properties that must be addressed through relationships. Relationships have also been identified as a promising concept for authorization specification in collaborative applications in other works [181]. However, as indicated earlier, properties such as the due date and whether an invoice was paid can not be modeled as straightforwardly through relationships, and hence benefits from attribute support in EBAC.

Lastly, EBAC enables better support for the principle of separation of concerns, because policies can be adapted without requiring changes in the underlying queries. Security administrators do not require cooperation with database experts, except for mapping the entity model onto the database. As such, the authorization specification effort is reduced, which curtails costs, lowers security risks involved with the cooperation, and shortens the time required for policy specification and adjustment.

5.6 Related work

Over the last decades, various models have been proposed to mitigate issues of prominent access control models such as RBAC [90]. This work is primarily inspired by Attribute-Based Access Control (ABAC, [120]) and Relationship-Based Access Control (ReBAC).

On the one hand, ABAC enables expressive policies, but does not support specification of complex relationships as was illustrated by rules 5.1 and 5.2. XACML [172] is considered the de-facto standard policy language for attribute-

based policies. This chapter introduces a policy language that was inspired by the policy model proposed by this standard. Section 5.3 introduced a policy language that supports the concepts such as policy trees, multi-valued logic, and combining algorithms that were key technologies adopted in XACML.

On the other hand, Relationship-Based Access Control (ReBAC, [93]) takes into account relationships in order to make access control decisions. This supports the expression of a great deal of rules, but does not naturally support attributes to be compared with each other. ReBAC was originally applied to social networks [51, 117, 136]. Crampton and Sellwood [70] extended this body of work by (a) arguing wider applicability of relationships than only social networks, and (b) considering entities other than subjects to be relevant in specifying relationships. Carminati et al. [51] analyzes the authorization requirements for social networks and introduces a model that restricts access based on type, depth and trust level of relationships. Moreover, they provided client-side authorization enforcement according to a rule-based approach. This work takes into account type and depth of relationships to determine privileges, which also inspired EBAC. Giunchiglia et al. proposed RelBAC [103], a model similar to ReBAC. Analogous to EBAC, the authors associate RelBAC to the Entity-Relationship model [58]. Fong proposed a ReBAC model that supports relationships that can be composed as a building block to build more complex relationships [93]. EBAC adapts a similar concept through vertical quantifier expressions, which is introduced in Section 5.3. Although all these models were influential to the conception of EBAC, a general issue for ReBAC is that it does not directly support comparisons of attributes of the involved entities, nor does it support intuitive temporal authorization constraints. Consequently, it does not always map seamlessly onto the application domain.

Of particular note for this contribution is the ReBAC model proposed by Crampton and Sellwood [70]. Relationships can be modeled in a *system model* as a bidirectional labeled graph G(V,E) in which the vertices $v \in V$ represent entities[2] and each edge $e \in E$ represents a relationship between two entities. Edges are assigned a label which reflects the type of relationship that exists between two connected entities. In order to specify policies in ReBAC, security administrators first specify a system model, which draws out all relevant types of entities and their possible relationships with each other. Based on this model, the security policy is specified by defining principal matching rules that designate relationship paths (called *path conditions*) of the system model onto roles (called *authorization principals*). Next, a list of authorization rules is described. Authorization rules explicitly assign permissions for resources and actions to authorization principals (or explicitly retract them).

[2]Prior ReBAC models only regard subjects as vertices [51, 136].

Recently, Ahmad et al. [6] have presented a comparative analysis of ABAC and ReBAC. In their work, they discussed how various features of ReBAC can be realized in ABAC. In particular, they discuss how composite relationship expressions can be adopted in ABAC through attribute chaining and composite attributes. While this work provides an interesting perspective on alternative approaches in which to adopt relationships in ABAC, they do not elaborate how complex expressions that leverage these relationships should be approached.

Besides access control models, EBAC was also inspired by several policy languages. SPL [193] is an influential policy language which supports entities and their relationships, comparison of properties and quantifiers. Similar to SPL, Ruby CanCan [21] supports addressing of entities and attribute comparisons. Because it is embedded in Ruby, CanCan is integrated seamlessly with the application model. Hachem et al. [108] introduced a policy framework for controlling access in mobile applications. This work featured, among others, a policy language which supported expressive policies. All these policy languages support reasoning about entities and relationships in a natural way. However, they do not support reasoning about relational quantifier expressions. Consequently, they do not entirely support EBAC. Moreover, the policy model introduced in this chapter supports policy modularization through tree-structured policies, which is generally not supported by the languages presented here.

Graph databases organize their data through models that support relationships, and support traversal of such relationships in data queries [10]. A subset of graph databases supports the nodes and edges to contain attributes that describe their properties. The corresponding query language may hence reflect the essence of entity-based languages as well. For example, Neo4j supports Cypher [189], which is a query language that can filter nodes based on both attributes and relationships.

Also relevant is the work of Verhanneman et al. [231]. Verhanneman et al. enforce separation of concerns by supporting organization-wide policies for which the rules are enforced in an aspect-oriented framework. This enables the security administrator to reason about application-specific concepts using Java code. However, changing such a policy requires recompiling the application code. The Drools [190] rules engine also enables application-specific concepts to be seamlessly integrated into business logic policies, at the expense of evaluation overhead.

5.7 Conclusion

This chapter presented Entity-Based Access Control (EBAC), an access control model that supports both comparison of attribute values, as well as traversing relationships of intermediate entities.

EBAC maps onto the application domain in an expressive way because its underlying ER-model can be applied to most application models. We have proposed a policy model and validated the feasibility to support EBAC by means of the Auctoritas policy language. The evaluation of Auctoritas indicates that (a) it supports relationships in a more expressive way than XACML and (b) that it introduces only limited performance overhead for policy evaluation. Support for these expressions involves no custom query specifications and hence reduce the management effort involved when such rules must be supported.

As such, EBAC increases expressiveness of the authorization policies. The benefit for this with regard to multi-organizational applications is threefold. First, it improves support for self-management, as security administrators can specify policies that align closer to their organizational structure. Second, EBAC maintains the principle of separation of concerns. Security administrators do not need to specify custom queries in order to reflect relationships that are implicit to the one-dimensional attributes that are supported by ABAC. This is especially beneficial to SaaS applications, because a need for such custom queries to accommodate security preferences may introduce security risks when performed on a shared infrastructure. Third, EBAC enables more flexible and precise sharing of resources between organizations in collaborative environments.

Consequently, we are convinced that EBAC provides a promising novel approach for more expressive authorization policy specification in multi-organizational applications.

Chapter 6

Conclusion

This chapter concludes the dissertation. Section 6.1 revisits the contributions and summarizes the requirements they address with regards to multi-organizational authorization. Section 6.3 presents some important future directions for access control in general, and multi-organizational authorization in particular. Section 6.4 discusses some concluding thoughts.

6.1 Contributions

The previous chapters of this thesis have addressed multi-organizational authorization management from the perspective of the organizations' security administrators, comprehensive enforcement, and expressiveness, respectively. As such, they addressed the core requirements that were outlined in Chapter 1. This section provides an overview of how each contribution addressed a subset of particular requirements.

Amusa. Our first contribution regarded management and enforcement of authorization in multi-tenant SaaS applications. Multi-tenant SaaS authorization requirements are equivalent to with those listed for multi-organizational applications. Amusa focused on isolation and empowered sharing through a multi-layered authorization management model that comprises both attribute and policy management. Amusa also enables self-management by means of a tenant administration dashboard. Self-management is supported through a tenant-specific management layer, which enables security administrators of tenants to specify their own authorization constraints and

indicate exceptions to the isolation for their own resources.

Amusa also provided a means to reduce performance overhead. This was accommodated through configurable performance tactics that included pushing and pulling attributes in the policy decision engine, and policy decomposition and evaluation at one of two tiers, thus making it a significant contribution to multi-organizational authorization.

Sequoia. While Amusa managed to enable multi-organizational authorization management and enforcement for control-focused operations (e.g., CRUD operations), it does not address these challenges in the context of *data-focused* operations that are performed on database systems (e.g., search queries).

Sequoia supports the aforementioned requirements for data-focused operations as well, because it enables authorization enforcement based on the same core technologies that constitute the basis of Amusa. As such, Sequoia addresses the *comprehensive enforcement* of authorization constraints. Through query rewriting based on dynamic run-time conditions, Sequoia injects the appropriate access rules into the original search or data aggregation queries in a way that the results contain only data items to which the acting subject is privileged.

EBAC. Lastly, the previous contributions were supported by attribute-based policies to determine subject privileges. These policies, however, leave a gap in terms of expressiveness, which can lead to an inability to manage access privileges based on organizational properties efficiently. This is problematic especially for multi-organizational applications. In particular, ABAC does not support the concept of relationships. For multi-organizational authorization, this complicates requirements such as self-management and sharing to a considerable extent because these frequently involve such concepts.

EBAC approaches this issue by treating entities as first-class citizens in authorization policies. Entities involve both attributes and relationships. In contrast to ABAC, auxiliary entities can be addressed in authorization policies through relationship paths starting from a subject, resource, action, or the environment. Similar to ABAC, EBAC determines privileges through comparison of entity attributes with each other or with concrete values.

This enables *organization-centric expressiveness* in authorization constraints. As such, EBAC provides three benefits to multi-organizational authorization. First, it improves self-management, because security administrators can align their policy specification closer to the organizational structure. Second, it supports the principle of separation of concerns, as there is no need for synthetic attributes that implicitly reflect relationships but require custom queries that must be specified by database experts to retrieve their values.

EBAC can generate these queries automatically because it embraces the concept of relationships. Third, it enables more flexible and precise sharing of resources between organizations. Consequently, EBAC contributes to authorization in multi-organizational applications through addressing some of their fundamental requirements.

Non-functional requirements. The contributions in this dissertation also addressed the cross-cutting non-functional requirements specified in Chapter 1. In particular, we addressed modifiability, centralization, reusability, and performance. First, through embracing core technologies such as policy-based access control, the contributions enabled run-time modifiability of authorization policies. This also addresses the demand for near-continuous up-time that is omnipresent in multi-organizational applications. Second, the contributions focused on centrally managed authorization. This reflects the need for mandatory access control administration that is determined and restricted by a central authority, i.e., the application provider. Third, reusability was supported for all contributions by encapsulating authorization functionality into a reusable middleware that can be used across multiple types of multi-organizational applications. Fourth, the evaluations of all contributions demonstrated that the performance overhead for accomplishing the requirements was limited. We expect that this overhead can be decreased even more through additional optimizations that were outside the scope of the contributions.

All contributions were motivated by several industry case studies which put forward the requirements that were discussed in this section. The case studies, discussed in Chapter 2, encompassed multiple application domains, in particular (i) electronic document processing, (ii) security monitoring services, (iii) workforce management, and (iv) electronic health care management. We therefore expect these requirements to be more generally applicable for authorization in a broad variety of multi-organizational applications.

6.2 Limitations

This dissertation has introduced three contributions which further the support for authorization in multi-organizational applications. While the approaches in this thesis improve management and enforcement support for multi-organizational authorization, they also have limitations. The reason for this is two-fold. On the one hand, any approach generally has limitations for application domains on which it can be applied [229]. On the other hand, there are inherent limitations for the contributions that apply to the security domain in general.

Federated authorization. Federated authorization support was not regarded for this thesis, but has been a point of attention in related work [11, 78]. While we believe that our contributions can complement a more federated form of authorization to some extent, pervasive support requires considerable challenges to be addressed first. Issues that must be addressed include, but are not limited to, external attribute management, policy decomposition, federated identity management, and trust management. Also, federated authorization may incur a significantly higher performance overhead for policy evaluations, which requires far-reaching performance optimizations to be put in place.

Discretionary management. One major assumption that was posited for this thesis, driven primarily by the case studies supported by our work, was the centrally managed nature of authorization administration. While our work supports authorization constraints and access rights to be self-managed by the involved organizations, we did not focus on delegation of such administrative rights. While related work has focused on this concept for collaborative environments (e.g., [130]), it typically focuses on role-based authorization systems. Future work should evaluate the opportunities and limitations of our work with regard to more discretionary management.

Legacy applications. Another limitation for our contributions is how they can be integrated to support legacy applications. These applications commonly have their own built-in access control subsystem in place, and integration with another access control system is complex, or sometimes even infeasible. For example, legacy applications that cannot be adapted can still be instrumented for authorization through an application firewall (e.g., [191]). However, policies could also require application-specific attributes that cannot be retrieved from a database prior to the execution of an action. This makes it impossible to perform full-fledged authorization, and could be prohibitive for the adoption of our contributions for such systems.

High-performance applications. Due to their dependency on technologies such as policy-based access control and attribute-based expressions, our contributions could also introduce a prohibitive performance overhead in high-performance application domains in which a low overhead is paramount (e.g., stock trading applications). Due to the nature of these applications, low-overhead authorization checks must be put in place instead, relying primarily on post-operation audit for more complex security policy enforcement.

Applicability. Support for a wide range of aspects such as workflow engine authorization, application-to-policy binding, and microservices, requires at least considerable extension of the contributions presented in this dissertation. For instance, the emerging concept of microservices aims for modularization of the application architecture through composition of a collection of loosely coupled

services (e.g., storage, caching, separate business operations) [57]. Microservices may require a more comprehensive policy management and enforcement approach. In such an approach, security-in-depth is ensured through performing authorization on both the overall action being performed (e.g., book an appointment), and each individual microservice that action requires (e.g., store the event in the database). This could be performed through policy refinement, but it is yet unclear to what extent this could impact our contributions and the granularity level at which microservice authorization can be performed. Moreover, this requires further research into the performance impact for authorization enforcement, and how to properly attribute a microservice request to the overall action in order to improve auditing.

Attack surface. This dissertation furthered the state-of-the-art by proposing extensions to existing access control approaches. Although this can improve authorization management, such extensions also require significant infrastructure extensions which broadens the attack surface that can be abused by malicious actors to gain unauthorized access. This is strongly related to the reference monitor model discussed in Chapter 2. Reference monitors should be tamper-proof, non-bypassible, and susceptible to verification methods [200]. However, practical authorization systems encompass multifarious requirements, putting in doubt whether the aforementioned goals for a reference monitor can ever feasibly be achieved for a realistic scenario. Nonetheless, continuous efforts must be made to pursue these goals to reduce the attack surface of the system as much as possible. Through formal verification of the policy transformation process in SEQUOIA and security analysis of policy composition in AMUSA, we did perform important steps to ensure this. However, additional efforts are still required to minimize the attack surface.

Security-in-Depth. Complementary security approaches can be employed to ensure security-in-depth. This includes management methodologies with extensive auditing efforts and separation of duty policies, and a general pursuit of the principle of least privileges [199] even for authorization management. Moreover, it includes data-level protection approaches such as attribute-based encryption [104], homomorphic encryption [101], and secure query processing [233]. While these technologies have challenges with regard to multi-organizational policies, dynamicity, and performance, they can, to some extent, be applied orthogonally to harden the security of software systems.

This (non-exhaustive) set of limitations indicates that a significant amount of future research is still required to increase the authorization capabilities in multi-organizational applications. With the contributions in this thesis, however, we have provided a meaningful step towards improved authorization support that can serve as a basis for future technologies.

6.3 Future directions

While this dissertation presented several contributions to authorization for multi-organizational applications, and to access control in general, a lot of research remains to be conducted. This section discusses future directions for access control that we deem important. In particular, we envision four major directions on which future research should focus: (i) policy specification on an organization-wide level, (ii) enhancing of authorization management with tooling and support for an integrated management approach, (iii) improving authorization enforcement coverage analysis and instrumentation, and (iv) adding intelligence, among others through machine learning techniques, to optimize authorization management.

6.3.1 Organization-wide policies

Firstly, we envision access control policies to increasingly evolve towards higher-level specifications. The goal of authorization policies has always revolved around restriction of users according to the organizational rules, culture, and high-level regulatory guidelines. Currently, this often comes down to interpreting the organizational policies into authorization rules for each operation on resources of each separate application. This daunting task is designated to security administrators that need to translate organizational policies into permissions, roles, attributes, and authorization rules for each separate application.

This is a time consuming and error-prone endeavor that would benefit from (partial) automation. This can be addressed through policy refinement. In order to perform policy refinement, high-level organizational policies that reason about business processes and responsibilities need to be translated to authorization policies for each application that is used within the organization. For example, consider expression 6.1, which reflects a high-level security policy.

"employees are never permitted, using any application, to change their own salary" (6.1)

Through refinement, such high-level policies can then be enforced for each application-specific operation. For example, authorization expression 6.1 can be refined for a database management application, that impedes users to modify their own salary through restrictions on the operations that the users can do on the relevant data fields. Similarly, the authorization policies for a human resources management application can also restrict specific operations that lead to salary changes.

Policy refinement supports translation of high-level operations to application-specific actions. Similarly, any high-level resource predicates must be translated to specific application-level attributes. This requires security administrators to provide a mapping that categorizes the resources and operations in order to project abstract high-level primitives onto the applications that require specific policies that must be enforced. This mapping enables a systematic refinement process that keeps the application access control consistent with the high-level policies.

The concept of policy refinement in order to abstract from application specifics has been a focus of research in the past [3, 20, 35, 231], and was the original goal for role-based access control [90]. However, this introduces a wide range of challenges. Among others, the granularity level and accommodation for flexible specification of authorization constraints on a high level must support refinement to the application level. In particular, access control systems of applications may not support the expressive rules that were specified on the higher level. Hence, there is a need for standardized models and interfaces, and mapping technologies between different access control models, to support the heterogeneity of authorization systems of applications. Moreover, there may be differing perspectives on the semantics of business operations and how they should be refined in application-specific authorization policies, which makes the ability to perform a proper mapping paramount for the approach. Such a mapping must serve as a guideline for the translation of concepts of organization-wide policies to application-specific policies. Additionally, in order for organization-wide policies to be effective, they should be pervasive, i.e., they should comprise every business operation, and as a consequence also every application operation, in order to restrain access to the organizational resources.

These challenges are by no means comprehensive, and it should be clear that this is rather a long-term research goal. However, this approach does provide a lot of benefits and opportunities. First, it intends to reduce the effort and errors with specification for security administrators, thus cutting costs. Second, it facilitates the adoption of a single, organization-wide reference monitor that performs the policy decision process for all refined application policies. Consequently, the monitoring of security properties and its enforcement performance can be streamlined so security administrators can respond to incidents more rapidly and effectively. Third, it presents a high-level, unified view of the privileges of users within an organization, and enables such view on resources that are shared by the organization with external actors (e.g., other organizations). Fourth, it enables the comprehensive auditing of the authorization system and policies. Fifth, it supports ancillary services, such as investigating whether every resource is accessible, or confirming whether the leave schedules of employees still allow for certain business operations to be performed (and authorized) at all times.

While this vision on access control calls for considerable longstanding research efforts, we believe that the aforementioned benefits make further research worthwhile.

6.3.2 An integrated management approach

Orthogonal to the previous research direction, there is also a need to improve the management capabilities of authorization policies in general. This encompasses a wide range of facets that would benefit from extensive research efforts, and are complicated especially by the emergence of more expressive access control models. We consider five such facets.

Application coupling. Closer research attention is required for support of tighter coupling of the authorization constraints with the application models about which they reason. This is especially true for more expressive access control models such as ABAC, ReBAC, and EBAC. Since these models encompass properties and/or relationships of the resources and operations (i.e., actions) of the application, the expressions are more prone to errors. For example, the expressions may compare properties that do not exist or represent another data type altogether.

Tighter application coupling can ensure that such errors can be detected at the specification time of a policy. Also, they can enable analysis on the *completeness* of a policy, i.e., detection of whether each resource type and application operation is reasoned about in the authorization policy. One approach to do this is through stronger type checking restrictions for access control policies based on an application model extracted semi-automatically from the application code.

Scalable management. Scalable management approaches are required to offload the burden of security administrators. We suggest two viable approaches for this, (i) privilege management guided by machine learning, which is outlined briefly further in this section, and (ii) decentralization of privilege assignments.

Decentralization fundamentally encompasses administrative delegation, and thus comes down to an approach to support some form of discretionary privilege management within a mandatory policy. This approach has been regarded extensively in research for RBAC models, in particular with *administrative* RBAC [202]. However, more expressive models can complicate this approach significantly. The multi-layered management approach that was presented in Amusa for inter-organizational management may address this issue for organizational departments and project teams as well. However, this requires further investigation to determine to what extent it conforms to

typical organizational structures and how it copes with deviations from such expectations.

Overhead minimization and policy analysis. More expressive policy models such as XACML impose greater performance overhead on the access control enforcement process [227]. Moreover, they complicate the ability to perform policy analysis [134]. While considerable amount of research has focused on performance optimization [146, 165, 168, 185] and policy analysis techniques [129, 134, 228], there still is a need for practical tools and approaches that enable security administrators to quickly analyze and evaluate the authorization policies they specify. This can increase the adoption of more expressive models, and ensure that sufficient auditing safeguards are in place for organizations.

In terms of performance optimization, a shift from authorization to after-the-fact audit for a part of the policy may also alleviate some issues [53]. This is the case especially for computationally expensive and otherwise heavy-overhead authorization rules. An authorization engine could, in accordance with the security administrator, evaluate certain rules *after* the action has been performed. As a result, these rules remain unenforced when the application performs the action. However, accountability for a user's actions is retained. This only applies for rules that are not compelled to stringent enforcement requirements, and builds on the concept of break-the-glass [92]. Users are (to some extent) trusted to access only resources to which they are privileged, and evaluation of expressive rules at audit time may cause the security administrators to be notified in case these rules were violated.

Expressiveness. Policy models such as EBAC focused on improving expressiveness, thereby enhancing the granularity at which privileges can be scalably specified. However, there are still some expressiveness gaps that should be addressed to facilitate the task of security administrators. Particularly, *deputyship rules* can enable constraints in which privileges are indirectly inherited in a different manner than role or attribute hierarchies. Consider a rule *"Nurses can read patients' medical records of the last month if their supervisor also has read privileges to these records"*. This rule implies that the authorization policy needs to be evaluated for the nurses' supervisors in order to determine whether the nurse can rightfully read it. Whereas attribute and role hierarchies can organize privileges if there is a logical structure that can be imposed on these properties, *deputyship* does not require such structure and enables independent adaptation of other users' privileges. This is somewhat similar to delegation, in which explicit permissions are transferred between users. However, deputyship rules are property-driven, instead of the often explicitly identity-driven delegation of privileges. Moreover, deputyship rules enable taking into account the evaluation context, and can compel further

restrictions. The XACML Administration and Delegation Profile [173] takes a similar, property-driven approach through chaining of policies that can grant additional privileges through trusted issuers. However, this approach lacks the dynamism provided by deputyship that enables policy evaluation *on behalf of* another user and takes into account that evaluation decision to determine privileges.

Integrated management. A single, integrated management approach can assimilate the aforementioned facets of application coupling, scalable management, expressiveness, and overhead minimization and policy analysis. An important observation that we have experienced from our contributions, is that increased expressiveness of policy models, such as XACML, can actually complicate the management process. Complex conflict resolution strategies and large sets of (possibly conflicting) rules may result in policies that yield different decisions than intended. This was also the main driver for policy analysis techniques [129, 134].

While an increased expressiveness thus can complicate the security administrators' management efforts to a significant extent, they also can provide vital management capabilities. Hence, research should provide the proper tools and methods to use these technologies in an optimal manner.

One approach to do this, paradoxically, is to *restrict* the security administrators in how they can organize their policies. For example, policies could be organized in multiple levels, that have expressions regarding different properties. A first level could regard the application actions, a second level the resource types, a third level the user roles, a fourth level another user attribute, and so on. An access request must apply to each level in order to be taken into account for an access decision. Such an approach may facilitate reasoning about conflicting rules, and could lead to less error-prone policy specification.

Orthogonally, a second approach is to encapsulate reusable authorization logic into policy trees, thus making policy specification a composition of reusable policy modules. Prior work has already regarded this in the form of policy templates [79], policy instantiation [73], and policy references [172]. However, further research is required to investigate to what extent such an approach can facilitate policy specification.

6.3.3 Enforcement coverage analysis

One major security issue, among others in web applications, is the misconfiguration of authorization enforcement points, leaving a part of the application operations unprotected [230]. In order to alleviate this issue, several research

efforts have been directed to automated detection and placement of authorization hooks [100, 166, 216, 220].

For instance, Sun et al. perform a static analysis based on the assumption that each actor performs a unique set of privileges on a web application, and that this is reflected in links shown in the presentation layer [220]. This work accesses these links through unprivileged roles to determine vulnerable web pages. Rolecast [216] takes a different approach that automatically infers the actions of an application, the security checks necessary for protecting security-sensitive operations, and detects missing security checks automatically. It does this through the observation that web applications only use a small number of data sources for authorization information (e.g., session state) that are checked through conditional structures, and help inference of existing sensitive operations. Muthukumaran et al. [166] identify and perform automatic hook placement for system applications through *constraint selectors* and recognition of language-specific operations (e.g., reading to file system). Through a mix of static and dynamic analysis techniques, they then detect an optimal hook placement. However, while these efforts can aid in the detection, they generally tend to focus on specific characteristics of the application code's language, domain, libraries, or other properties to address the issue. Also, it is unclear whether these approaches detect every operation that needs to be authorized.

There is a need for approaches that can facilitate the detection of such unprotected operations, or otherwise ensure that no publicly facing application operations (i.e., to which users directly make requests) can be performed without access control. Such approaches could be embedded in the programming languages or frameworks, or alternatively could comprise middleware layers that intercept all requests that users make to the application. While industry efforts already provide such approaches to some extent, security analysis statistics imply that it still remains a considerable problem [230]. Hence, research must address the shortcomings of these approaches in order to ensure that authorization enforcement is performed comprehensively.

6.3.4 Management and enforcement optimization through machine learning

Over the last years, a lot of research attention has been drawn onto leveraging machine learning for security applications. This has yielded interesting results, among others for detecting malicious domain name registrations [111]. In terms of access control, authentication has leveraged such techniques through continuous authentication by monitoring user behavior [183]. For authorization, these techniques have been applied for role [138] and attribute [237] mining, and

in network-level intrusion detection systems [14]. Generally, machine learning could be employed to facilitate addressing some key challenges of authorization with regard to management and enforcement. This can be leveraged to reduce management costs and operational overhead. Moreover, it can minimize policy decision processing, thereby improving performance of the policy evaluation.

Machine learning could enable addressing fundamental management challenges through optimization of administrative operations such as access right management and anomaly detection based on the set of existing permission assignments, or based on historical access data. Similarly, machine learning could also further reduce policy evaluation overhead based on trend analysis or predict future access attempts based on such historical data. While a plethora of approaches can be performed for such optimizations, we elaborate on examples in (i) application-level anomaly detection, (ii) authorization management, and (iii) policy evaluation optimization to provide a more concrete idea of the opportunities enabled by machine learning.

Application-level anomaly detection. Intrusion detection has been extensively studied for network-level access control. This concept enables security administrators to detect suspicious network behavior and perform appropriate and timely responses [14]. Similarly, Security Information and Event Management (SIEM) systems have focused on the aggregation, correlation, and reporting of security events based on audit trails [30]. Given proper support for audit capabilities and instrumentation, further research of machine learning and deep learning techniques performed on application-level audit trails can improve anomaly detection effectiveness. This would allow a more granular analysis of incidents since application operations could be correlated and used to restrict suspicious users in a certain time frame.

Authorization management. Management of authorization can also benefit from applying machine learning techniques, as it could enable more scalable privilege management. Through correlating user attributes and their privileges, any gaps in permissions (or over-assigned permissions) could be detected. For example, machine learning could be applied to detect that users of a certain department and with a certain role typically have a certain restriction, and that another user's privileges deviate from this. This could be used by security administrators to perform policy audits. Correlating attributes from users with similar characteristics may also facilitate authorization management for the security administrator and reduce costs, as it could indicate suggestions for attribute assignments based on users with similar responsibilities. A similar research field to this is role mining [138], however, more expressive policy models considerably complicate this issue and thus require further research.

Policy evaluation optimization. Lastly, the policy evaluation process could

also be optimized through machine learning. For instance, attribute retrieval time comprises a significant portion of the total evaluation overhead [45]. Machine learning techniques could be applied to determine, for certain authorization requests, which attributes will likely be required in the policy evaluation process. These can then be pushed alongside the authorization request for more timely responses. Another example of where machine learning may yield performance optimization is in adaptive policy organization. In this approach, common authorization requests are correlated according to certain time frames in order to restructure authorization policies to decrease policy evaluation overhead. This has already been applied in a static way in previous research [155]. However, some applications have periodic access peaks, which may comprise different common requests. For example, in document processing applications, pay checks are typically generated at a certain time period on a monthly basis, whereas invoices are likely generated in larger amounts in some periods in the year (e.g., Christmas holidays). Such peaks are not addressed by this previous work, but could be applied here as well. For AMUSA, such techniques could also be employed to determine policy decomposition and deployment on the cooperating PDPs according to which rules are typically decisive in the policy evaluation process, and require only minimal attribute retrieval requests.

Machine learning may also aid in improving federated authorization architectures to support decisions that are made more locally. In particular, employing machine learning in decentralized architectures for (partially) performing local authorization at autonomous sites based on historical authorization decisions can reduce attribute retrieval operations, reduce the required trust level to some extent, and increase performance. The extent to which this can be applied is an interesting direction for future research, as it may significantly change how decentralized authorization architectures operate as well.

6.4 Concluding thoughts

This dissertation focused on advancing authorization support for multi-organizational applications. In particular, we have addressed several important requirements that were introduced in Chapter 1, such as *isolation*, *sharing*, *self-management*, *comprehensive enforcement*, and *improved expressiveness* in this context.

With a continuously increasing focus on collaboration and openness in computer software, support for such requirements is becoming paramount. Software delivery models such as Software-as-a-Service (SaaS), an important

category of multi-organizational applications, are drawing an increasing interest. Consequently, these trends are driving authorization technologies and research to support environments in which organizations are separate, isolated, independent entities. These entities demand authorization support for common resources, and want to specify their own authorization policies aligned to their organizational structure. Moreover, resource sharing is increasingly being mandated by industry regulations. These regulations are continuously evolving and driving requirements such as access by third-party audit organizations and governmental bodies. As a result, multi-organizational software systems are becoming increasingly prevalent. This introduces novel challenges that require systematic solutions.

The contributions that were presented in this thesis address an important part of these challenges. As such, we have improved support for multi-organizational authorization through significant contributions to the state-of-the art. Hence, these contributions have the potential to enable an increased adoption of multi-organizational applications, and by extension further access control in general.

Bibliography

[1] Abadi, M. Logic in Access Control. In *Logic in Computer Science, 2003. Proceedings. 18th Annual IEEE Symposium on* (2003), IEEE, pp. 228–233.

[2] Abdunabi, R., Al-Lail, M., Ray, I., and France, R. B. Specification, Validation, and Enforcement of a Generalized Spatio-Temporal Role-Based Access Control Model. *IEEE Systems Journal 7*, 3 (2013), pp. 501–515.

[3] Abrams, M., and Bailey, D. Abstraction and Refinement of Layered Security Policy. *Information Security: An Integrated Collection of Essays* (1995), pp. 126–136.

[4] Ahmadian, M., Plochan, F., Roessler, Z., and Marinescu, D. C. SecureNoSQL: An Approach for Secure Search of Encrypted NoSQL Databases in the Public Cloud. *International Journal of Information Management 37*, 2 (2017), pp. 63–74.

[5] Ahmed, T., Patwa, F., and Sandhu, R. Object-to-Object Relationship-Based Access Control: Model and Multi-Cloud Demonstration. In *Information Reuse and Integration (IRI), 2016 IEEE 17th International Conference on* (2016), IEEE, pp. 297–304.

[6] Ahmed, T., Sandhu, R., and Park, J. Classifying and Comparing Attribute-Based and Relationship-Based Access Control. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy* (2017), ACM, pp. 59–70.

[7] Alam, M., Zhang, X., Khan, K., and Ali, G. xDAuth: A Scalable and Lightweight Framework for Cross Domain Access Control and Delegation. In *Proceedings of the 16th ACM symposium on Access control models and technologies* (2011), ACM, pp. 31–40.

[8] Alcaraz Calero, J. M., Edwards, N., Kirschnick, J., Wilcock, L., and Wray, M. Toward a Multi-Tenancy Authorization System for Cloud Services. *IEEE Security & Privacy 8*, 6 (2010), pp. 48–55.

[9] Alfieri, R., Cecchini, R., Ciaschini, V., dell'Agnello, L., Frohner, A., Gianoli, A., Lorentey, K., and Spataro, F. VOMS, an Authorization System for Virtual Organizations. In *Grid computing* (2004), Springer, pp. 33–40.

[10] Angles, R. A Comparison of Current Graph Database Models. In *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on* (2012), IEEE, pp. 171–177.

[11] Ardagna, C. A., di Vimercati, S. D. C., Neven, G., Paraboschi, S., Preiss, F.-S., Samarati, P., and Verdicchio, M. Enabling Privacy-preserving Credential-based Access Control with XACML and SAML. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on* (2010), IEEE, pp. 1090–1095.

[12] Ashley, P., Hada, S., Karjoth, G., Powers, C., and Schunter, M. Enterprise Privacy Authorization Language (EPAL). Tech. rep., IBM Research, 2003.

[13] Ashley, P. M., and Vandenwauver, M. *Practical Intranet Security: Overview of the State of the Art and Available Technologies.* Springer Science & Business Media, 1999.

[14] Axelsson, S. Intrusion Detection Systems: A Survey and Taxonomy. Tech. rep., Chalmers University of Technology, 2000.

[15] Axiomatics. Data Access Filter. `https://axiomatics.com/product/axiomatics-data-access-filter-adaf/`, March 2018.

[16] Azeez, A., Perera, S., Gamage, D., Linton, R., Siriwardana, P., Leelaratne, D., Weerawarana, S., and Fremantle, P. Multi-Tenant SOA Middleware for Cloud Computing. In *Cloud computing (cloud), 2010 ieee 3rd international conference on* (2010), IEEE, pp. 458–465.

[17] Bacon, J., Evans, D., Eyers, D. M., Migliavacca, M., Pietzuch, P., and Shand, B. Enforcing End-to-End Application Security in the Cloud. In *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware* (2010), Springer-Verlag, pp. 293–312.

[18] Banerjee, S. P., and Woodard, D. L. Biometric Authentication and Identification using Keystroke Dynamics: A survey. *Journal of Pattern Recognition Research 7*, 1 (2012), pp. 116–139.

[19] Baracaldo, N., Masoumzadeh, A., and Joshi, J. A Secure, Constraint-Aware Role-Based Access Control Interoperation Framework.

In *Network and System Security (NSS), 2011 5th International Conference on* (2011), IEEE, pp. 200–207.

[20] Basin, D., Doser, J., and Lodderstedt, T. Model Driven Security: From UML Models to Access Control Infrastructures. *ACM Trans. Softw. Eng. Methodol. 15*, 1 (Jan. 2006), pp. 39–91.

[21] Bates, R. Ruby CanCan. `https://www.rubydoc.info/gems/cancan/`, March 2018.

[22] Becker, M. Y., Fournet, C., and Gordon, A. D. SecPAL: Design and Semantics of a Decentralized Authorization Language. *Journal of Computer Security 18*, 4 (2010), pp. 619–665.

[23] Becker, M. Y., and Sewell, P. Cassandra: Distributed Access Control Policies with Tunable Expressiveness. In *Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on* (2004), IEEE, pp. 159–168.

[24] Bell, D. E., and La Padula, L. J. Secure computer systems. Tech. rep., Mitre Corporation, 1976. Mitre TR-2547.

[25] Benantar, M. *Access Control Systems: Security, Identity Management and Trust Models*. Springer Science & Business Media, 2006.

[26] Bertino, E., and Sandhu, R. Database Security – Concepts, Approaches, and Challenges. *Dependable and Secure Computing, IEEE Transactions on 2*, 1 (2005), pp. 2–19.

[27] Bertolino, A., Daoudagh, S., Lonetti, F., and Marchetti, E. Automatic XACML Requests Generation for Policy Testing. In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on* (2012), IEEE, pp. 842–849.

[28] Bertolissi, C., and Fernández, M. A Rewriting Framework for the Composition of Access Control Policies. In *Proceedings of the 10th international ACM SIGPLAN conference on Principles and practice of declarative programming* (2008), ACM, pp. 217–225.

[29] Bezemer, C.-P., Zaidman, A., Platzbeecker, B., Hurkmans, T., and Hart, A. Enabling Multi-Tenancy: An Industrial Experience Report. In *Software Maintenance (ICSM), 2010 IEEE International Conference on* (2010), IEEE, pp. 1–8.

[30] Bhatt, S., Manadhata, P. K., and Zomlot, L. The Operational Role of Security Information and Event Management Systems. *IEEE Security Privacy 12*, 5 (September 2014), pp. 35–41.

[31] Bhatti, R., Ghafoor, A., Bertino, E., and Joshi, J. B. X-gtrbac: an xml-based policy specification framework and architecture for enterprise-wide access control. *ACM Transactions on Information and System Security (TISSEC) 8*, 2 (2005), pp. 187–227.

[32] Biba, K. J. Integrity Considerations for Secure Computer Systems. Tech. rep., Mitre Corporation, 1977. Mitre TR-3153.

[33] Biswas, P., Sandhu, R., and Krishnan, R. Label-Based Access Control: An ABAC Model with Enumerated Authorization Policy. In *Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control* (2016), ACM, pp. 1–12.

[34] Bogaerts, J., Decat, M., Lagaisse, B., and Joosen, W. Entity-Based Access Control: Supporting More Expressive Access Control Policies. In *Proceedings of the 31st Annual Computer Security Applications Conference* (2015), ACM, pp. 291–300.

[35] Bogaerts, J., and Lagaisse, B. Improving Manageability of Access Control Policies. In *ESSoS Doctoral Symposium* (2014).

[36] Bogaerts, J., Lagaisse, B., and Joosen, W. Idea: Supporting Policy-Based Access Control on Database Systems. In *International Symposium on Engineering Secure Software and Systems* (2016), Springer, pp. 251–259.

[37] Bogaerts, J., Lagaisse, B., and Joosen, W. SEQUOIA: Scalable Policy-Based Access Control for Search Operations in Data-Driven Applications. In *International Symposium on Engineering Secure Software and Systems* (2017), Springer, pp. 1–18.

[38] Bogaerts, J., Lagaisse, B., and Joosen, W. Transforming XACML policies into database search queries. Tech. rep., KU Leuven, 2017.

[39] Bogaerts, J., Lagaisse, B., and Joosen, W. SEQUOIA: a middleware supporting policy-based access control for search and aggregation in data-driven applications. *Transactions on Dependable and Secure Computing* (2018). Submitted – in review.

[40] Boutaba, R., and Aib, I. Policy-Based Management: A Historical Perspective. *Journal of Network and Systems Management 15*, 4 (2007), pp. 447–480.

[41] Boyer, J. P., Hasan, R., Olson, L. E., Borisov, N., Gunter, C. A., and Raila, D. Improving Multi-tier Security using Redundant Authentication. In *Proceedings of the 2007 ACM workshop on Computer security architecture* (2007), ACM, pp. 54–62.

[42] Brewer, D. F., and Nash, M. J. The Chinese Wall Security Policy. In *Security and privacy, 1989. proceedings., 1989 ieee symposium on* (1989), IEEE, pp. 206–214.

[43] Brostoff, S., Sasse, M. A., Chadwick, D., Cunningham, J., Mbanaso, U., and Otenko, S. 'R-What?' Development of a Role-Based Access Control Policy-Writing Tool for e-Scientists. *Software: Practice and Experience 35*, 9 (2005), pp. 835–856.

[44] Brucker, A. D., and Petritsch, H. Extending Access Control Models with Break-Glass. In *Proceedings of the 14th ACM symposium on Access control models and technologies* (2009), ACM, pp. 197–206.

[45] Brucker, A. D., and Petritsch, H. Idea: Efficient Evaluation of Access Control Constraints. In *International Symposium on Engineering Secure Software and Systems* (2010), Springer, pp. 157–165.

[46] Bruns, G., Fong, P. W., Siahaan, I., and Huth, M. Relationship-Based Access Control: Its Expression and Enforcement through Hybrid Logic. In *Proceedings of the second ACM conference on Data and Application Security and Privacy* (2012), ACM, pp. 117–124.

[47] Caetano, A., Silva, A. R., and Tribolet, J. A Role-Based Enterprise Architecture Framework. In *Proceedings of the 2009 ACM symposium on Applied Computing* (2009), ACM, pp. 253–258.

[48] Carminati, B., and Ferrari, E. Privacy-Aware Collaborative Access Control in Web-Based Social Networks. In *IFIP Annual Conference on Data and Applications Security and Privacy* (2008), Springer, pp. 81–96.

[49] Carminati, B., and Ferrari, E. Collaborative Access Control in Online Social Networks. In *Collaborative computing: networking, applications and worksharing (CollaborateCom), 2011 7th international conference on* (2011), IEEE, pp. 231–240.

[50] Carminati, B., Ferrari, E., Cao, J., and Tan, K. L. A Framework to Enforce Access Control over Data Streams. *ACM TISSEC* (2010).

[51] Carminati, B., Ferrari, E., and Perego, A. Rule-Based Access Control for Social Networks. In *OTM Meaningful Internet Systems* (2006), Springer.

[52] Carrie, D., and Gates, E. Access Control Requirements for Web 2.0 Security and Privacy. In *Proc. of Workshop on Web 2.0 Security & Privacy (W2SP 2007* (2007), vol. 2.

[53] Cederquist, J. G., Corin, R., Dekker, M. A. C., Etalle, S., den Hartog, J. I., and Lenzini, G. Audit-Based Compliance Control. *International Journal of Information Security 6*, 2 (Mar 2007), pp. 133–151.

[54] Chadwick, D., Otenko, A., and Ball, E. Role-Based Access Control with X.509 Attribute Certificates. *IEEE Internet Computing 7*, 2 (2003), pp. 62–69.

[55] Chadwick, D., Zhao, G., Otenko, S., Laborde, R., Su, L., and Nguyen, T. A. PERMIS: A Modular Authorization Infrastructure. *Concurrency and Computation: Practice and Experience 20*, 11 (2008), pp. 1341–1357.

[56] Chatvichienchai, S., Iwaihara, M., and Kambayashi, Y. Secure Interoperability between Cooperating XML Systems by Dynamic Role Translation. In *DEXA: International conference on database and expert systems applications* (Prague , Tcheque Republique, 2003), pp. 866–875.

[57] Chen, L. Microservices: Architecting for Continuous Delivery and DevOps. In *IEEE International Conference on Software Architecture (ICSA)* (2018).

[58] Chen, P. P.-S. The Entity-Relationship Model – Toward a Unified View of Data. *Readings in Artificial Intelligence and Databases* (1989), pp. 98 – 111.

[59] Cheng, Y., Park, J., and Sandhu, R. A User-to-User Relationship-Based Access Control Model for Online Social Networks. In *IFIP Annual Conference on Data and Applications Security and Privacy* (2012), Springer, pp. 8–24.

[60] Chong, F., Carraro, G., and Wolter, R. Multi-Tenant Data Architecture. Tech. rep., Microsoft Corporation, 2006. MSDN Library.

[61] Clark, D. D., and Wilson, D. R. A Comparison of Commercial and Military Computer Security Policies. In *Security and Privacy, 1987 IEEE Symposium on* (1987), IEEE, pp. 184–184.

[62] Colombo, M., Lazouski, A., Martinelli, F., and Mori, P. Access and Usage Control in Grid Systems. In *Handbook of Information and Communication Security*. Springer, 2010, pp. 293–308.

[63] Colombo, P., and Ferrari, E. Enhancing MongoDB with Purpose Based Access Control. *IEEE Transactions on Dependable and Secure Computing* (2015).

[64] Colombo, P., and Ferrari, E. Towards virtual private NoSQL datastores. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on* (2016), IEEE, pp. 193–204.

[65] Colombo, P., and Ferrari, E. Towards a Unifying Attribute Based Access Control Approach for NoSQL Datastores. In *IEEE 33rd Conference on Data Engineering* (2017).

[66] Columbus, L. Cloud Computing Market Projected To Reach $411B By 2020. *Forbes* (2017).

[67] Concord, Inc. Concord: the contract success platform. `http://www.concordnow.com/`. Accessed: 2018-01-11.

[68] Contract Room, Inc. Contactroom: discover the power of data-driven negotiation and contracting. `http://contractroom.com`. Accessed: 2018-01-11.

[69] Crampton, J., and Khambhammettu, H. Delegation in Role-Based Access Control. *International Journal of Information Security 7*, 2 (Apr 2008), pp. 123–136.

[70] Crampton, J., and Sellwood, J. Path Conditions and Principal Matching: A New Approach to Access Control. In *Proceedings of the 19th ACM symposium on Access control models and technologies* (2014), ACM, pp. 187–198.

[71] Damen, S., den Hartog, J., and Zannone, N. CollAC: Collaborative Access Control. In *Collaboration Technologies and Systems (CTS), 2014 International Conference on* (2014), IEEE, pp. 142–149.

[72] Damiani, M. L., Bertino, E., Catania, B., and Perlasca, P. GEO-RBAC: a Spatially Aware RBAC. *ACM Transactions on Information and System Security (TISSEC) 10*, 1 (2007), pp. 2.

[73] Damianou, N., Dulay, N., Lupu, E., and Sloman, M. The Ponder Policy Specification Language. In *Policies for Distributed Systems and Networks*. Springer, 2001, pp. 18–38.

[74] De Win, B., Piessens, F., Joosen, W., and Verhanneman, T. On the importance of the separation-of-concerns principle in secure software engineering. In *Workshop on the Application of Engineering Principles to System Security Design* (2002), pp. 1–10.

[75] Decat, M., Bogaerts, J., Lagaisse, B., and Joosen, W. The e-document case study: functional analysis and access control requirements. Tech. rep., Department of Computer Science, KU Leuven, 2014.

[76] Decat, M., Bogaerts, J., Lagaisse, B., and Joosen, W. The workforce management case study: functional analysis and access control requirements. Tech. rep., Department of Computer Science, KU Leuven, 2014.

[77] Decat, M., Bogaerts, J., Lagaisse, B., and Joosen, W. Amusa: Middleware for Efficient Access Control Management of Multi-Tenant SaaS Applications. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing* (2015), ACM, pp. 2141–2148.

[78] Decat, M., Lagaisse, B., Van Landuyt, D., Crispo, B., and Joosen, W. Federated Authorization for Software-as-a-Service Applications. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"* (2013), Springer, pp. 342–359.

[79] Decat, M., Moeys, J., Lagaisse, B., and Joosen, W. Improving Reuse of Attribute-Based Access Control Policies Using Policy Templates. In *International Symposium on Engineering Secure Software and Systems* (2015), Springer, pp. 196–210.

[80] Dekker, M., Crampton, J., and Etalle, S. RBAC Administration in Distributed Systems. In *Proceedings of the 13th ACM symposium on Access control models and technologies* (2008), ACM, pp. 93–102.

[81] Dell, Inc. End-User Security Survey, September 2017. [Online; posted September-2017; visited 06-February-2018].

[82] Dijkstra, E. W. *A Discipline of Programming*, vol. 1. Prentice-Hall Englewood Cliffs, 1976.

[83] Elasticsearch BV. Elasticsearch. `https://www.elastic.co/`, March 2018.

[84] Elasticsearch BV. Elasticsearch Java API. `https://www.elastic.co/guide/en/elasticsearch/client/java-api/current/`, March 2018.

[85] Elliott, A., and Knight, S. Role Explosion: Acknowledging the Problem. In *Software Engineering Research and Practice* (2010), pp. 349–355.

[86] Ellison, C. RFC2692 – SPKI Requirements. RFC, IETF, 1999.

[87] Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.-G., Cox, L. P., Jung, J., McDaniel, P., and Sheth, A. N. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring

on Smartphones. *ACM Transactions on Computer Systems (TOCS) 32*, 2 (2014), pp. 5.

[88] Engine, G. A. Namespaces API for Java. `https://cloud.google.com/appengine/docs/standard/java/multitenancy/`. Accessed: 2018-03-06.

[89] Fatema, K., Chadwick, D. W., and Lievens, S. A Multi-Privacy Policy Enforcement System. In *IFIP PrimeLife International Summer School on Privacy and Identity Management for Life* (2010), Springer, pp. 297–310.

[90] Ferraiolo, D. F., Sandhu, R., Gavrila, S., Kuhn, D. R., and Chandramouli, R. Proposed NIST Standard for Role-Based Access Control. *ACM Transactions on Information and System Security (TISSEC) 4*, 3 (2001), pp. 224–274.

[91] Ferrari, E. Access Control in Data Management Systems. *Synthesis lectures on data management 2*, 1 (2010), pp. 1–117.

[92] Ferreira, A., Cruz-Correia, R., Antunes, L., Farinha, P., Oliveira-Palhares, E., Chadwick, D. W., and Costa-Pereira, A. How To Break Access Control in a Controlled Manner. In *Computer-Based Medical Systems, 2006. CBMS 2006. 19th IEEE International Symposium on* (2006), IEEE, pp. 847–854.

[93] Fong, P. W. Relationship-Based Access Control: Protection Model and Policy Language. In *Proceedings of the first ACM conference on Data and application security and privacy* (2011), ACM, pp. 191–202.

[94] Foster, I., Kesselman, C., Tsudik, G., and Tuecke, S. A Security Architecture for Computational Grids. In *Proceedings of the 5th ACM conference on Computer and communications security* (1998), ACM, pp. 83–92.

[95] Foster, I., Zhao, Y., Raicu, I., and Lu, S. Cloud Computing and Grid Computing 360-Degree Compared. In *Grid Computing Environments Workshop, 2008. GCE'08* (2008), Ieee, pp. 1–10.

[96] Frank, M., Biedert, R., Ma, E., Martinovic, I., and Song, D. Touchalytics: On the Applicability of Touchscreen Input as a Behavioral Biometric for Continuous Authentication. *IEEE transactions on information forensics and security 8*, 1 (2013), pp. 136–148.

[97] Franzoni, S., Mazzoleni, P., Valtolina, S., and Bertino, E. Towards a Fine-Grained Access Control Model and Mechanisms

for Semantic Databases. In *Web Services, 2007. ICWS 2007. IEEE International Conference on* (2007), IEEE, pp. 993–1000.

[98] Freier, A., Karlton, P., and Kocher, P. The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC, IETF, 2011.

[99] Fuchs, L., Pernul, G., and Sandhu, R. Roles in Information Security – A Survey and Classification of the Research Area. *computers & security 30*, 8 (2011), pp. 748–769.

[100] Ganapathy, V., King, D., Jaeger, T., and Jha, S. Mining Security-Sensitive Operations in Legacy Code Using Concept Analysis. In *Software Engineering, 2007. ICSE 2007. 29th International Conference on* (2007), IEEE, pp. 458–467.

[101] Gentry, C. Fully Homomorphic Encryption Using Ideal Lattices. In *STOC* (2009), vol. 9, pp. 169–178.

[102] Giambiagi, P. Abbreviated Language for Authorization (ALFA) Version 1.0. https://www.oasis-open.org/committees/download.php/55228/alfa-for-xacml-v1.0-wd01.doc, 2015.

[103] Giunchiglia, F., Zhang, R., and Crispo, B. Relbac: Relation Based Access Control. In *Semantics, Knowledge and Grid* (2008), IEEE.

[104] Goyal, V., Pandey, O., Sahai, A., and Waters, B. Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data. In *Proceedings of the 13th ACM conference on Computer and communications security* (2006), Acm, pp. 89–98.

[105] Grassi, P., Garcia, M., and Fenton, J. SP 800-63-3 - Digital Identity Guidelines. *NIST Special Publication* (2017).

[106] Grubb, S. Auditctl – A Utility To Assist Controlling the Kernel's Audit System. https://linux.die.net/man/8/auditctl. Accessed: 2018-08-13.

[107] Guo, C. J., Sun, W., Huang, Y., Wang, Z. H., and Gao, B. A Framework for Native Multi-Tenancy Application Development and Management. In *The 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (CEC-EEE 2007)* (July 2007), pp. 551–558.

[108] Hachem, S., Toninelli, A., Pathak, A., and Issarny, V. Policy-Based Access Control in Mobile Social Ecosystems. In *Policies for Distributed Systems and Networks (POLICY)* (2011), IEEE.

[109] Han, J., Haihong, E., Le, G., and Du, J. Survey on NoSQL Database. In *Pervasive computing and applications (ICPCA), 2011 6th international conference on* (2011), IEEE, pp. 363–366.

[110] Han, W., and Lei, C. A Survey on Policy Languages in Network and Security Management. *Computer Networks 56*, 1 (2012), pp. 477–489.

[111] Hao, S., Kantchelian, A., Miller, B., Paxson, V., and Feamster, N. PREDATOR: Proactive Recognition and Elimination of Domain Abuse at Time-Of-Registration. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), ACM, pp. 1568–1579.

[112] Hardt, D. The OAuth 2.0 Authorization Framework. RFC, IETF, 2012.

[113] Harrison, M. A., Ruzzo, W. L., and Ullman, J. D. Protection in Operating Systems. *Communications of the ACM 19*, 8 (1976), pp. 461–471.

[114] Hasebe, K., Mabuchi, M., and Matsushita, A. Capability-Based Delegation Model in RBAC. In *Proceedings of the 15th ACM Symposium on Access Control Models and Technologies* (2010), pp. 109–118.

[115] Hecht, R., and Jablonski, S. NoSQL Evaluation: A Use Case Oriented Survey. In *Cloud and Service Computing (CSC), 2011 International Conference on* (2011), IEEE, pp. 336–341.

[116] Hoelzer, D. A SANS Survey on Continuous Monitoring, 2015. [Online; posted 2015; visited 20-November-2017].

[117] Hu, H., Ahn, G.-J., and Jorgensen, J. Multiparty Access Control for Online Social Networks: Model and Mechanisms. *IEEE Transactions on Knowledge and Data Engineering 25*, 7 (2013), pp. 1614–1627.

[118] Hu, H., Chen, D., and Huang, C. Securing Role-Based Distributed Collaboration System. *IEEE International Conference on Systems, Man and Cybernetics 6* (2004), pp. 5520– 5524.

[119] Hu, V. C., Ferraiolo, D., Kuhn, R., Friedman, A. R., Lang, A. J., Cogdell, M. M., Schnitzer, A., Sandlin, K., Miller, R., Scarfone, K., et al. Guide to Attribute Based Access Control (ABAC): Definition and Considerations. *NIST special publication 800*, 162 (January 2014).

[120] Hu, V. C., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., Miller, R., and Scarfone, K. Guide to Attribute Based Access

Control (ABAC) Definition and Considerations. *NIST Special Publication* (2014).

[121] Ilia, P., Carminati, B., Ferrari, E., Fragopoulou, P., and Ioannidis, S. SAMPAC: Socially-Aware Collaborative Multi-Party Access Control. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy* (2017), ACM, pp. 71–82.

[122] Impetus. Kundera. `https://www.impetus.com/content/jpa-compliant-polyglot-nosql-client-and-object-mapper-kundera-201`, March 2018.

[123] Internet2. Shibboleth – Privacy Preserving Identity Management. `https://www.shibboleth.net/`. Accessed: 2018-02-27.

[124] ISO. ISO/IEC 9075-2:2016, Information technology – Database languages – SQL – Part 2: Foundation (SQL/Foundation). `https://www.iso.org/standard/63556.html`, 2016.

[125] Jahid, S., Gunter, C. A., Hoque, I., and Okhravi, H. MyABDAC: Compiling XACML Policies for Attribute-Based Database Access Control. In *Proceedings of the first ACM conference on Data and application security and privacy* (2011), ACM, pp. 97–108.

[126] Jajodia, S., Gadia, S., and Bhargava, G. Logical Design of Audit Information in Relational Databases. *Information Security: An integrated Collection of Essays* (1995), pp. 585–595.

[127] Jajodia, S., Samarati, P., and Subrahmanian, V. A Logical Language for Expressing Authorizations. In *Security and Privacy, 1997. Proceedings., 1997 IEEE Symposium on* (1997), IEEE, pp. 31–42.

[128] Jammalamadaka, R. C., Gamboni, R., Mehrotra, S., Seamons, K. E., and Venkatasubramanian, N. iDataGuard: Middleware Providing a Secure Network Drive Interface to Untrusted Internet Data Storage. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology* (2008), ACM, pp. 710–714.

[129] Jebbaoui, H., Mourad, A., Otrok, H., and Haraty, R. Semantics-Based Approach for Detecting Flaws, Conflicts and Redundancies in XACML Policies. *Computers & Electrical Engineering 44* (2015), pp. 91–103.

[130] Jin, J., and Ahn, G.-J. Role-Based Access Management for Ad-Hoc Collaborative Sharing. In *Proceedings of the eleventh ACM symposium on Access control models and technologies* (2006), ACM, pp. 200–209.

[131] Jin, X., Krishnan, R., and Sandhu, R. A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC. In *IFIP Annual Conference on Data and Applications Security and Privacy* (2012), Springer, pp. 41–55.

[132] Jin, X., Sandhu, R., and Krishnan, R. RABAC: Role-Centric Attribute-Based Access Control. In *International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security* (2012), Springer, pp. 84–96.

[133] Karp, A. H. Authorization-Based Access Control for the Services Oriented Architecture. In *Creating, Connecting and Collaborating through Computing, 2006. C5'06. The Fourth International Conference on* (2006), IEEE, pp. 160–167.

[134] Kolovski, V., Hendler, J., and Parsia, B. Analyzing Web Access Control Policies. In *Proceedings of the 16th International Conference on World Wide Web* (New York, NY, USA, 2007), WWW07, ACM, pp. 677–686.

[135] Krebs, R., Momm, C., and Kounev, S. Architectural Concerns in Multi-tenant SaaS Applications. *Closer 12* (2012), pp. 426–431.

[136] Kruk, S. R., Grzonkowski, S., Gzella, A., Woroniecki, T., and Choi, H.-C. D-FOAF: Distributed Identity Management with Access Rights Delegation. In *The Semantic Web – ASWC 2006*. Springer, 2006.

[137] Kuhn, D. R., Coyne, E. J., and Weil, T. R. Adding Attributes to Role-Based Access Control. *Computer 43*, 6 (2010), pp. 79–81.

[138] Kumar, R., Sural, S., and Gupta, A. Mining RBAC Roles Under Cardinality Constraint. In *International Conference on Information Systems Security* (2010), Springer, pp. 171–185.

[139] Kumar, V., Cooper, B. F., Eisenhauer, G., and Schwan, K. iManage: Policy-Driven Self-Management for Enterprise-Scale Systems. In *Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware* (2007), Springer-Verlag New York, Inc., pp. 287–307.

[140] Lambert, S. 2018 SaaS Industry Market Report: Key Global Trends & Growth Forecasts. *FinancesOnline* (2018).

[141] Lampson, B. W. Protection. *ACM SIGOPS Operating Systems Review 8*, 1 (1974), pp. 18–24.

[142] Lazouski, A., Mancini, G., Martinelli, F., and Mori, P. Usage Control in Cloud Systems. In *Internet Technology And Secured Transactions, 2012 International Conference for* (2012), IEEE, pp. 202–207.

[143] Li, Q., Zhang, X., Xu, M., and Wu, J. Towards Secure Dynamic Collaborations with Group-Based RBAC Model. *Computers & Security 28*, 5 (2009), pp. 260–275.

[144] Liang, F., Guo, H., Yi, S., and Ma, S. A Multiple-Policy supported Attribute-Based Access Control Architecture within Large-scale Device Collaboration Systems. *JNW 7*, 3 (2012), pp. 524–531.

[145] Lin, D., Rao, P., Bertino, E., Li, N., and Lobo, J. Policy Decomposition for Collaborative Access Control. In *Proceedings of the 13th ACM symposium on Access control models and technologies* (2008), ACM, pp. 103–112.

[146] Liu, A. X., Chen, F., Hwang, J., and Xie, T. Xengine: A Fast and Scalable XACML Policy Evaluation Engine. In *ACM SIGMETRICS Performance Evaluation Review* (2008), vol. 36, ACM, pp. 265–276.

[147] Liu, A. X., Chen, F., Hwang, J., and Xie, T. Designing Fast and Scalable XACML Policy Evaluation Engines. *IEEE Transactions on Computers 60*, 12 (2011), pp. 1802–1817.

[148] Longstaff, J., and Noble, J. Attribute Based Access Control for Big Data Applications by Query Modification. In *Big Data Computing Service and Applications (BigDataService), 2016 IEEE Second International Conference on* (2016), IEEE, pp. 58–65.

[149] Lorch, M., Adams, D. B., Kafura, D., Koneni, M., Rathi, A., and Shah, S. The PRIMA System for Privilege Management, Authorization and Enforcement in Grid Environments. In *Proceedings of the 4th International Workshop on Grid Computing* (2003), IEEE Computer Society, p. 109.

[150] Lu, H., Vaidya, J., and Atluri, V. Optimal Boolean Matrix Decomposition: Application to Role Engineering. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on* (2008), IEEE, pp. 297–306.

[151] Lu, Y., Zhang, L., Liu, Y., and Sun, J. A Distributed Domain Administration of RBAC Model in Collaborative Environments. In *CSCWD: International Conference on Computer Supported Cooperative Work in Design* (2006), pp. 935–940.

[152] Lunt, T. F., Denning, D. E., Schell, R. R., Heckman, M., and Shockley, W. R. The SeaView Security Model. *IEEE Transactions on software engineering 16*, 6 (1990), pp. 593–607.

[153] Lyu, M. R., and Lau, L. K. Y. Firewall Security: Policies, Testing and Performance Evaluation. In *Proceedings 24th Annual International Computer Software and Applications Conference. COMPSAC2000* (2000), pp. 116–121.

[154] Mainka, C., Mladenov, V., Feldmann, F., Krautwald, J., and Schwenk, J. Your Software at my Service: Security Analysis of SaaS Single Sign-On Solutions in the Cloud. In *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security* (2014), ACM, pp. 93–104.

[155] Marouf, S., Shehab, M., Squicciarini, A., and Sundareswaran, S. Adaptive Reordering and Clustering-Based Framework for Efficient XACML Policy Evaluation. *IEEE Transactions on Services Computing 4*, 4 (2011), pp. 300–313.

[156] Martin, E., and Xie, T. Automated Test Generation for Access Control Policies via Change-Impact Analysis. In *Proceedings of the Third International Workshop on Software Engineering for Secure Systems* (2007), IEEE Computer Society, p. 5.

[157] Mazzoleni, P., Crispo, B., Sivasubramanian, S., and Bertino, E. XACML Policy Integration Algorithms. *ACM Transactions on Information and System Security (TISSEC) 11*, 1 (2008), pp. 4.

[158] Mazzoleni, P., Crispo, B., Sivasubramanian, S., and Bertino, E. Efficient Integration of Fine-Grained Access Control and Resource Brokering in Grid. *The Journal of Supercomputing 49*, 1 (2009), pp. 108.

[159] McCarty, B. *SELinux: NSA's Open Source Security Enhanced Linux*, vol. 238. O'Reilly, 2005.

[160] Medvet, E., Bartoli, A., Carminati, B., and Ferrari, E. Evolutionary Inference of Attribute-Based Access Control Policies. In *International Conference on Evolutionary Multi-Criterion Optimization* (2015), Springer, pp. 351–365.

[161] Mell, P., Grance, T., et al. The NIST Definition of Cloud Computing. Tech. rep., National Institute of Standards and Technology, 2011. Special Publication 800-145.

[162] Menzel, M., Wolter, C., and Meinel, C. Access Control for Cross-Organisational Web Service Composition. *Journal of Information Assurance and Security 2*, 3 (2007), pp. 155–160.

[163] Modi, C., Patel, D., Borisaniya, B., Patel, H., Patel, A., and Rajarajan, M. A Survey of Intrusion Detection Techniques in Cloud. *Journal of Network and Computer Applications 36*, 1 (2013), pp. 42–57.

[164] Moore, S. Gartner Says Worldwide Information Security Spending Will Grow 7 Percent to Reach $ 86.4 Billion in 2017, August 2017. [Online; posted 16-August-2017; visited 4-October-2017].

[165] Mourad, A., and Jebbaoui, H. SBA-XACML: Set-Based Approach Providing Efficient Policy Decision Process for Accessing Web Services. *Expert Systems with Applications 42*, 1 (2015), pp. 165–178.

[166] Muthukumaran, D., Jaeger, T., and Ganapathy, V. Leveraging Choice to Automate Authorization Hook Placement. In *Proceedings of the 2012 ACM conference on Computer and communications security* (2012), ACM, pp. 145–156.

[167] Mutti, S., Bacis, E., and Paraboschi, S. SeSQLite: Security Enhanced SQLite: Mandatory Access Control for Android databases. In *Proceedings of the 31st Annual Computer Security Applications Conference* (2015), ACM, pp. 411–420.

[168] Ngo, C., Demchenko, Y., and de Laat, C. Decision Diagrams for XACML Policy Evaluation and Management. *Computers & Security 49* (2015), pp. 1–16.

[169] Ni, Q., and Bertino, E. xfACL: An Extensible Functional Language for Access Control. In *Proceedings of the 16th ACM symposium on Access control models and technologies* (2011), ACM, pp. 61–72.

[170] Ni, Q., Lobo, J., Calo, S., Rohatgi, P., and Bertino, E. Automating Role-Based Provisioning by Learning from Examples. In *Proceedings of the 14th ACM symposium on Access control models and technologies* (2009), ACM, pp. 75–84.

[171] OASIS. Security Assertion Markup Language (SAML) v2.0. `https://www.oasis-open.org/standards#samlv2.0`, 2005.

[172] OASIS. eXtensible Access Control Markup Language (XACML) Standard, Version 3.0. `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf`, 2013.

[173] OASIS. XACML v3.0 Administration and Delegation Profile Version 1.0. http://docs.oasis-open.org/xacml/3.0/administration/v1.0/xacml-3.0-administration-v1.0.pdf, 2014.

[174] Olson, L. E., Gunter, C. A., Cook, W. R., and Winslett, M. Implementing Reflective Access Control in SQL. In *Data and Applications Security*. Springer, 2009.

[175] Ong, K. W., Papakonstantinou, Y., and Vernoux, R. The SQL++ Unifying Semi-Structured Query Language, and an Expressiveness Benchmark of SQL-on-Hadoop, NoSQL and NewSQL Databases. *CoRR, abs/1405.3631* (2014).

[176] Opyrchal, L., Cooper, J., Poyar, R., Lenahan, B., and Zeinner, D. Bouncer: Policy-Based Fine Grained Access Control in Large Databases. *Itl. Journal of Security and Its Applications* (2011).

[177] Oracle. JDBC. https://www.oracle.com/technetwork/java/overview-141217.html, March 2018.

[178] Oracle. JPA. https://oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html, March 2018.

[179] Oracle. Virtual Private Database. https://docs.oracle.com/cd/B28359_01/network.111/b28531/vpd.htm, March 2018.

[180] Osborn, S., Sandhu, R., and Munawer, Q. Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies. *ACM Transactions on Information and System Security (TISSEC) 3*, 2 (2000), pp. 85–106.

[181] Paci, F., Squicciarini, A., and Zannone, N. Survey on Access Control for Community-Centered Collaborative Systems. *ACM Computing Surveys (CSUR) 51*, 1 (2018), pp. 6.

[182] Parnas, D. L. On the Criteria To Be Used in Decomposing Systems into Modules. *Communications of the ACM 15*, 12 (1972), pp. 1053–1058.

[183] Patel, V. M., Chellappa, R., Chandra, D., and Barbello, B. Continuous User Authentication on Mobile Devices: Recent Progress and Remaining Challenges. *IEEE Signal Processing Magazine 33*, 4 (2016), pp. 49–61.

[184] Pathirage, M., Perera, S., Kumara, I., and Weerawarana, S. A Multi-Tenant Architecture for Business Process Executions. In *Web services (icws), 2011 ieee international conference on* (2011), IEEE, pp. 121–128.

[185] Pina Ros, S., Lischka, M., and Gómez Mármol, F. Graph-Based XACML Evaluation. In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies* (2012), ACM, pp. 83–92.

[186] Pustchi, N., and Sandhu, R. MT-ABAC: A Multi-Tenant Attribute-Based Access Control Model with Tenant Trust. In *International Conference on Network and System Security* (2015), Springer, pp. 206–220.

[187] Rafique, A., Van Landuyt, D., and Joosen, W. PERSIST: Policy-Based Data Management Middleware for Multi-Tenant SaaS Leveraging Federated Cloud Storage. *Journal of Grid Computing* (2018), pp. 1–30.

[188] RedisLabs. Redis. `https://redis.io/`, March 2018.

[189] Neo4j, I. Neo4j's Graph Query Language: An Introduction to Cypher. `https://neo4j.com/developer/cypher-query-language/`, March 2018.

[190] Red Hat, I. Drools - Business Rules Management System. `https://www.drools.org/`, March 2018.

[191] Trustwave. ModSecurity Open Source Web Application Firewall. `https://www.modsecurity.org/`, July 2018.

[192] Riad, K., Yan, Z., Hu, H., and Ahn, G.-J. AR-ABAC: A New Attribute Based Access Control Model Supporting Attribute-Rules for Cloud Computing. In *Collaboration and Internet Computing (CIC), 2015 IEEE Conference on* (2015), IEEE, pp. 28–35.

[193] Ribeiro, C., Zuquete, A., Ferreira, P., and Guedes, P. SPL: An Access Control Language for Security Policies and Complex Constraints. In *NDSS* (2001), vol. 1.

[194] Rissanen, E. Fine-Grained Relational Database Access-Control Policy Enforcement Using Reverse Queries, May 2015. US Patent 9,037,610.

[195] Rizvi, S., Mendelzon, A., Sudarshan, S., and Roy, P. Extending Query Rewriting Techniques for Fine-Grained Access Control. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data* (2004), ACM, pp. 551–562.

[196] Roichman, A., and Gudes, E. Fine-grained access control to web databases. In *SACMAT* (2007), ACM.

[197] Ron, A., Shulman-Peleg, A., and Puzanov, A. Analysis and Mitigation of NoSQL Injections. *IEEE Security & Privacy 14*, 2 (2016), pp. 30–39.

[198] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and Mortimore, C. OpenID Connect Core 1.0. `http://openid.net/specs/openid-connect-core-1_0.html`, 2014.

[199] Saltzer, J. H., and Schroeder, M. D. The Protection of Information in Computer Systems. *Proceedings of the IEEE 63*, 9 (1975), pp. 1278–1308.

[200] Samarati, P., and Vimercati, S. Access Control: Policies, Models, and Mechanisms. In *Foundations of Security Analysis and Design*. Springer Berlin Heidelberg, 2001, pp. 137–196.

[201] Sandhu, R. The Authorization Leap from Rights to Attributes: Maturation or Chaos? In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies* (2012), ACM, pp. 69–70.

[202] Sandhu, R., Bhamidipati, V., and Munawer, Q. The ARBAC97 Model For Role-Based Administration of Roles. *ACM Transactions on Information and System Security (TISSEC) 2*, 1 (1999), pp. 105–135.

[203] Sandhu, R., and Samarati, P. Authentication, Access Control, and Audit. *ACM Computing Surveys (CSUR) 28*, 1 (1996), pp. 241–243.

[204] Sandhu, R. S. Lattice-Based Access Control Models. *Computer 26*, 11 (1993), pp. 9–19.

[205] Sandhu, R. S., and Samarati, P. Access Control: Principles and Practice. *IEEE communications magazine 32*, 9 (1994), pp. 40–48.

[206] Schaad, A., Moffett, J., and Jacob, J. The Role-Based Access Control System of a European Bank: A Case Study and Discussion. In *SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies* (New York, NY, USA, 2001), ACM, pp. 3–9.

[207] Schläger, C., Sojer, M., Muschall, B., and Pernul, G. Attribute-Based Authentication and Authorisation Infrastructures for E-Commerce Providers. In *International Conference on Electronic Commerce and Web Technologies* (2006), Springer, pp. 132–141.

[208] Servos, D., and Osborn, S. L. HGABAC: Towards a Formal Model of Hierarchical Attribute-Based Access Control. In *International Symposium on Foundations and Practice of Security* (2014), Springer, pp. 187–204.

[209] Servos, D., and Osborn, S. L. Current Research and Open Problems in Attribute-Based Access Control. *ACM Computing Surveys (CSUR) 49*, 4 (2017), pp. 65.

[210] Shafiq, B., Joshi, J. B., Bertino, E., and Ghafoor, A. Secure Interoperation in a Multidomain Environment Employing RBAC Policies. *IEEE transactions on knowledge and data engineering 17*, 11 (2005), pp. 1557–1577.

[211] Shehab, M., Bertino, E., and Ghafoor, A. SERAT: SEcure Role mApping Technique for decentralized secure interoperability. In *SACMAT* (2005), pp. 159–167.

[212] Shroff, G. *Enterprise Cloud Computing: Technology, Architecture, Applications*. Cambridge University Press, 2010.

[213] Sinnott, R. O., Chadwick, D. W., Doherty, T., Martin, D., Stell, A., Stewart, G., Su, L., and Watt, J. Advanced Security for Virtual Organizations: The Pros and Cons of Centralized vs Decentralized Security Models. In *Cluster Computing and the Grid, 2008. CCGRID'08. 8th IEEE International Symposium on* (2008), IEEE, pp. 106–113.

[214] Sloman, M. Policy Driven Management for Distributed Systems. *Journal of network and Systems Management 2*, 4 (1994), pp. 333–360.

[215] Smari, W. W., Clemente, P., and Lalande, J.-F. An Extended Attribute Based Access Control Model with Trust and Privacy: Application to a Collaborative Crisis Management System. *Future Generation Computer Systems 31* (2014), pp. 147–168.

[216] Son, S., McKinley, K. S., and Shmatikov, V. Rolecast: Finding Missing Security Checks When You Do Not Know What Checks Are. In *ACM SIGPLAN Notices* (2011), vol. 46, ACM, pp. 1069–1084.

[217] Spring Security. Expression-Based Access Control. `https://docs.spring.io/spring-security/site/docs/3.0.x/reference/el-access.html`, March 2018.

[218] Stoller, S. D., Yang, P., Ramakrishnan, C. R., and Gofman, M. I. Efficient Policy Analysis for Administrative Role Based Access Control. In *Proceedings of the 14th ACM conference on Computer and communications security* (2007), ACM, pp. 445–455.

[219] Stonebraker, M., and Wong, E. Access Control in a Relational Data Base Management System By Query Modification. In *Proceedings of the 1974 annual conference-Volume 1* (1974), ACM, pp. 180–186.

[220] Sun, F., Xu, L., and Su, Z. Static Detection of Access Control Vulnerabilities in Web Applications. In *USENIX Security Symposium* (2011).

[221] Sun, W., Zhang, X., Guo, C. J., Sun, P., and Su, H. Software as a Service: Configuration and Customization Perspectives. In *Congress on Services Part II, 2008. SERVICES-2. IEEE* (2008), IEEE, pp. 18–25.

[222] Suoranta, S., Manzoor, K., Tontti, A., Ruuskanen, J., and Aura, T. Logout in single sign-on systems: Problems and solutions. *Journal of Information Security and Applications 19*, 1 (2014), pp. 61–77.

[223] Tang, B., Sandhu, R., and Li, Q. Multi-Tenancy Authorization Models for Collaborative Cloud Services. *Concurrency and Computation: Practice and Experience 27*, 11 (2015), pp. 2851–2868.

[224] The Apache Software Foundation. Cassandra Wide-Column Store. `https://cassandra.apache.org/`, March 2018.

[225] The MariaDB Foundation. MariaDB. `https://mariadb.org/`, March 2018.

[226] Tripathi, A. R., Ahmed, T., and Kumar, R. Specification of Secure Distributed Collaboration Systems. In *Autonomous Decentralized Systems, 2003. ISADS 2003. The Sixth International Symposium on* (2003), IEEE, pp. 149–156.

[227] Turkmen, F., and Crispo, B. Performance Evaluation of XACML PDP Implementations. In *Proceedings of the 2008 ACM workshop on Secure web services* (2008), ACM, pp. 37–44.

[228] Turkmen, F., den Hartog, J., Ranise, S., and Zannone, N. Formal analysis of XACML policies using SMT. *Computers & Security 66* (2017), pp. 185 – 203.

[229] Uzunov, A. V., Fernandez, E. B., and Falkner, K. Security Solution Frames and Security Patterns for Authorization in Distributed, Collaborative Systems. *Computers & Security 55* (2015), pp. 193–234.

[230] van der Stock, A., Glas, B., Smithline, N., and Gigler, T. OWASP Top 10 - 2017, The Ten Most Critical Web Application Security Risks, November 2017. [Online; posted 20-November-2017; visited 08-January-2018].

[231] Verhanneman, T., Piessens, F., De Win, B., and Joosen, W. Uniform Application-Level Access Control Enforcement of Organizationwide Policies. In *Computer Security Applications Conference, 21st Annual* (2005), IEEE, pp. 10–pp.

[232] Vollbrecht, J., Calhoun, P., Farrell, S., Gommans, L., Gross, G., de Bruijn, B., de Laat, C., Holdrege, M., and Spence, D. AAA Authorization Framework. RFC, IETF, August 2000.

[233] Wang, S., Agrawal, D., and El Abbadi, A. A Comprehensive Framework for Secure Query Processing on Relational Data in the Cloud. In *Secure Data Management*. Springer, 2011, pp. 52–69.

[234] Wichers, D., Manico, J., Seil, M., and Mishra, D. SQL Injection Prevention Cheat Sheet. `https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet`, March 2018.

[235] Wun, A., and Jacobsen, H.-A. A Policy Management Framework for Content-Based Publish/Subscribe Middleware. In *Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware* (2007), Springer-Verlag New York, Inc., pp. 368–388.

[236] Xu, D., Thomas, L., Kent, M., Mouelhi, T., and Le Traon, Y. A Model-Based Approach to Automated Testing of Access Control Policies. In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies* (2012), ACM, pp. 209–218.

[237] Xu, Z., and Stoller, S. D. Mining Attribute-Based Access Control Policies from RBAC Policies. In *Emerging Technologies for a Smarter World (CEWIT), 2013 10th International Conference and Expo on* (2013), IEEE, pp. 1–6.

[238] Xu, Z., and Stoller, S. D. Mining Attribute-Based Access Control Policies . *IEEE Transactions on Dependable and Secure Computing 12*, 5 (2015), pp. 533–545.

[239] Yang, P., Gofman, M. I., Stoller, S. D., and Yang, Z. Policy Analysis for Administrative Role-Based Access Control Without Separate Administration. *Journal of Computer Security 23*, 1 (2015), pp. 1–29.

[240] Younis, Y. A., Kifayat, K., and Merabti, M. An Access Control Model for Cloud Computing. *Journal of Information Security and Applications 19*, 1 (2014), pp. 45–60.

[241] Yuan, E., and Tong, J. Attributed Based Access Control (ABAC) for Web Services. In *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on* (2005), IEEE.

[242] Zhang, X., Oh, S., and Sandhu, R. PBDM: A Flexible Delegation Model in RBAC. In *Proceedings of the eighth ACM symposium on Access control models and technologies* (2003), ACM, pp. 149–157.

[243] Zhang, Z., Zhang, X., and Sandhu, R. ROBAC: Scalable Role and Organization Based Access Control Models. In *Collaborative Computing: Networking, Applications and Worksharing, 2006. CollaborateCom 2006. International Conference on* (2006), IEEE, pp. 1–9.

[244] Zhao, D., Qiao, K., and Raicu, I. HyCache+: Towards Scalable High-Performance Caching Middleware for Parallel File Systems. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on* (2014), IEEE, pp. 267–276.

[245] Zhu, J., and Smari, W. W. Attribute Based Access Control and Security for Collaboration Environments. In *Aerospace and Electronics Conference, 2008. NAECON 2008. IEEE National* (2008), IEEE, pp. 31–35.

[246] Zuo, Q., Xie, M., and Tsai, W.-T. Autonomous Decentralized Authorization and Authentication Management for Hierarchical Multi-Tenancy. *IEICE Transactions on Communications 99*, 4 (2016), pp. 786–793.

List of publications

- Bogaerts, J., Decat, M., Lagaisse, B. and Joosen, W., 2015, December. Entity-Based Access Control: supporting more expressive access control policies. In Proceedings of the 31st Annual Computer Security Applications Conference (pp. 291-300). ACM.
- Decat, M., Bogaerts, J., Lagaisse, B. and Joosen, W., 2015, April. Amusa: middleware for efficient access control management of multi-tenant SaaS applications. In Proceedings of the 30th Annual ACM Symposium on Applied Computing (pp. 2141-2148). ACM.
- Bogaerts, J., Lagaisse, B. and Joosen, W., 2016, April. Idea: Supporting Policy-Based Access Control on Database Systems. In International Symposium on Engineering Secure Software and Systems (pp. 251-259). Springer, Cham.
- Bogaerts, J., Lagaisse, B. and Joosen, W., 2017, July. SEQUOIA: Scalable Policy-Based Access Control for Search Operations in Data-Driven Applications. In International Symposium on Engineering Secure Software and Systems (pp. 1-18). Springer, Cham.
- Bogaerts, J. and Lagaisse, B., 2014, February. Improving Manageability of Access Control Policies. In ESSoS Doctoral Symposium.
- Bogaerts, J., Lagaisse, B. and Joosen, W., 2017. Transforming XACML policies into database search queries. Technical report.
- Decat, M., Bogaerts, J., Lagaisse, B. and Joosen, W., 2014. The e-document case study: functional analysis and access control requirements. Technical report.
- Decat, M., Bogaerts, J., Lagaisse, B. and Joosen, W., 2014. The workforce management case study: functional analysis and access control requirements. Technical report.

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
IMEC-DISTRINET
Celestijnenlaan 200A box 2402
B-3001 Leuven
jasper.bogaerts@cs.kuleuven.be
https://distrinet.cs.kuleuven.be