The present work was submitted to the Research Group Software Construction

of the Faculty of Mathematics, Computer Science, and Natural Sciences

Master Thesis

Automation of security policies in software systems

presented by

Merzough Badry Münker

Aachen, December 18, 2023

Examiner

Prof. Dr. rer. nat. Horst Lichter

Prof. Dr. rer. nat. Bernhard Rumpe

Supervisor

Alex Sabau, M.Sc.

Acknowledgment

Firstly, I extend my thanks to my supervisor, Alex Sabau, for the opportunity to delve into this fascinating subject under his guidance. His support and valuable insights have been instrumental in shaping this thesis.

I would also like to express my gratitude to Prof. Dr. rer. nat. Horst Lichter for his constructive feedback and academic guidance. His expertise and encouragement have been greatly appreciated. Additionally, my thanks go to Prof. Dr. rer. nat. Bernhard Rumpe for his role in reviewing this thesis as the secondary examiner.

A special acknowledgment to my family, whose unwavering support and belief in my capabilities have been a constant source of strength throughout my studies. To my parents and my brother, thank you for your encouragement and support.

Lastly, I must thank my friends for their companionship and support, which have been a great comfort to me during this journey. Your presence and encouragement have been invaluable.

Thank you.

Merzough Badry Münker

Abstract

As our world increasingly digitizes, the security of software systems becomes paramount. There exists a wide variety of security policies and their modeling approaches. Existing security policy models, however, often lack accessibility for stakeholders without deep domain knowledge or include multiple aspects irrelevant to some stakeholders. Furthermore, these security policy models predominantly automate security policies based on the interests of a limited subset of stakeholders, consequently overlooking the concerns of others. This thesis addresses this gap by defining the concepts of Security Policy and Automation to determine the elements of Security Policies that are amenable to automation. In response, we introduce the Security Policy Meta Model (SPMM), designed to enable the creation of tailored Security Policy Models (SPM) that resonate with stakeholder concerns. Additionally, the SPMM incorporates an Automation Model (AM), which automates the modeled Security Policy, culminating in the Security Policy Automation Model (SPAM). Our methodology involves conducting a lightweight SLR to establish an overview of the current approaches of modeling security policy and there automation. Followed by developing a structured process for constructing a tailored SPAM. This process is validated through an application example, examining its applicability across multiple security policies and stakeholder groups. The findings from the application example highlight the SPMM's effectiveness in enhancing stakeholder comprehension of security policies and in facilitating the tailored automation of these policies. The establishment of the SPMM, coupled with its construction process, marks a significant advancement in simplifying the complexity of security policies for a broad spectrum of stakeholders and in progressing the field of security policy automation within software systems.

Contents

List of Tables

List of Figures

List of Source Codes

1. Introduction

Contents

In a world that has become increasingly interconnected with digital software systems [Hag+18], security policies are necessary to ensure the security of our digital world [Rou+23]. In general, a security policy is a set of rules and procedures that dictate how a software system should operate to ensure data integrity, confidentiality, and availability, and to prevent unauthorized access and misuse [10].

In the development of software systems, it is important to incorporate security policies at every stage of the *Software Development Life Cycle (SDLC)*. Integrating security policies from the outset is essential to ensure that they are an integral part of the software system, rather than a mere afterthought [Rou+23; NC16].

Given the requirement to recognize security policies at every stage of the SDLC, it is essential that all stakeholders involved in the software project understand the security policies. It is not sufficient for only those stakeholders with technical background to participate in the development and implementation of security policies [NC16; PTN09]. Consequently, there is a clear need to model security policies tailored to the concerns of stakeholders in order to improve their understanding of security policies [RKG22]. To improve the integration of security policies into the SDLC and minimize the risk of human error, the importance of security policy automation is increasingly recognized [AE08b; HBM07].

Automation can take several forms, including code generation, automated detection of security policy violations, and supporting tools for software system development. One approach is the use of formal methods to formally verify and capture the satisfaction relationship between security policies and security objectives, as proposed by Rouland et al. [Rou+23]. This allows automated analysis of the relationship between selected security solutions and security concerns in the software architecture design. Another approach is to use automated testing of network security controls. Here, network devices generate network traffic to validate the compliance of the protected network with desired security controls. Then it is assessed how network security policies process that traffic, as described by Hastings et al. [Rou+23]. In addition, Martínez et al. [14] propose a reverse engineering and integration mechanism for access control policies using model-driven technologies. This allows for automated discovery and understanding

of security policies. Another approach is to implement a security policy enforcement mechanism during the development phase of software applications. This mechanism allows security policies for data fields to be easily incorporated into transactions, providing flexibility and collaboration between programmers, domain experts, and security experts [NC16]. In addition, model-driven technologies can be used to reverse engineer and integrate access control policies, bridging the gap between vendor-dependent security features and a higher-level representation that is easier to understand and manipulate [14]. Furthermore, data security systems can use mappings to make security policies data-agnostic, portable, reusable, multifunctional, and comprehensive [13]. Finally, a software engineering approach can be used to automate the definition and enforcement of security policies in distributed systems [HBM07].

Current approaches to automation focus primarily on security policy automation from the perspective of technical stakeholders, with an emphasis on supporting security policy implementation. As a result, there is a noticeable gap in modeling security policy automation that addresses the needs and perspectives of all stakeholders. Existing modeling approaches primarily support stakeholders with a technical background. However, since security policies must be considered in every phase of the SDLC [Rou+23; NC16], it is critical that all stakeholders involved in software development understand these security policies [RKG22]. Additionally, different stakeholders may have different requirements for security policy automation, which requires the modeling of multiple automation solutions for the same security policy. Finally, depending on stakeholder concerns, it may be necessary to model only certain aspects of a security policy rather than the entire security policy.

1.1. Research questions

To address this perceived gap, three research questions have been identified to help bridge the gap. The first question involves investigating the structure of security policies to understand how they are organized and how their key aspects can be identified. The second question focuses on exploring how these aspects of security policies can be tailored to address the concerns of different stakeholders. Finally, the third question relates to investigating the automation of security policies in software systems.

(RQ1) - *What are the key aspects of security policies in software systems, and how are they structured?*

This research question aims to dissect the structure of security policies in software systems, focusing on identifying their key aspects and understanding the interrelationships among these key aspects. It will provide a fundamental understanding of security policies that is critical for both technical and non-technical stakeholders.

(RQ2) - *How can the aspects of security policies in software systems be tailored towards the concerns of stakeholders?*

This research question focuses on bridging the gap between the technical aspects of security policies and the diverse concerns of stakeholders. It explores strategies for adapting security policies to improve their accessibility, understanding, and relevance to different stakeholder groups.

(RQ3) - *In what sense can the aspects of security policies in software systems be automated and what are the challenges of automating them?*

This research question investigates the prospects of using security policy automation. It will explore possible approaches to automation, assess their feasibility, and identify challenges.

1.2. Goals and contributions

The goal of this thesis is to establish a foundation to bridge the gap in existing modeling approaches. Thus, facilitating the construction of security policy models and their automation tailored to address the concerns of all stakeholders. This thesis contributes to bridging this gap by proposing a *Security Policy Meta Model (SPMM)* and two concept models. The first concept model, the *Security Policy Concept Model (SPCM)*, assists in the construction of security policy models tailored to stakeholder concerns. The second concept model, the *Security Policy Automation Concept Model (SPACM)*, builds on the SPCM to facilitate the construction of models for the automation of security policy. Finally, this thesis presents a comprehensive process that provides structured guidance for constructing models consistent with the SPCM and the SPACM.

1.3. Research Methodology

In order to achieve our defined goal, we have designed the following research methodology. We use a lightweight *Systematic Literature Review (SLR)* to gain a fundamental understanding of security policies in software systems. A lightweight SLR adheres to the structured approach of a traditional SLR, but is tailored to be more concise to fit the scope of a master's thesis. While maintaining the systematic, transparent, and reproducible nature of an SLR, it involves a more focused selection of literature sources and may streamline certain steps to balance thoroughness with feasibility. This approach allows for an overview of the current state of research while ensuring manageability within the broader context of developing an SPMM and evaluation. This methodology is particularly suited to our objectives, as it provides a rigorous yet pragmatic basis for the subsequent development of the SPMM and Concept Models. Next, we will develop a construction process. This construction process is designed to evaluate the applicability of the SPMM and Concept Models using application examples. The creation of these application examples will also serve to refine the SPMM and Concept Models.

1.4. Structure of this Thesis

The structure of this thesis is as follows:

- In Chapter 3, the foundations of the terms *security policy in software systems* and *automation of security policy* are established, providing a common understanding for the following chapters.
- Chapter 2 presents related work to provide a clear understanding of existing approaches and contributions to modeling security policies in software systems. This chapter identifies strengths and limitations in the current landscape, thereby justifying the need for the proposed SPMM and concept models, and illustrating how they aim to address existing gaps.
- In Chapter 4 the process and results of the lightweight systematic literature review are detailed. The insights gained from this review directly inform the development of the SPMM and Concept Models.
- Chapter 5 introduces the SPCM, a key element of our proposed SPMM. This chapter demonstrates how the SPCM facilitates the construction of SPMs that are tailored to different stakeholder concerns, directly addressing one of the key goals of this research.
- Chapter 6 introduces the SPACM, a key element of our proposed SPMM. This chapter focuses on how the SPACM helps model security policy automation and complements the SPMM.
- Chapter 7 introduces the *Construction Process (CP)* for building SPCM and SPACM models. This chapter is critical because it provides a structured process for applying the SPMM.
- In Chapter 8 we evaluate the SPMM using three different application examples. This chapter is critical to validating the practical applicability and effectiveness of the SPMM and Concept Models. Each example illustrates how the models can be used in different contexts, demonstrating their versatility and relevance to real-world software system security scenarios.
- Chapter 9 discusses the findings from the application examples and assesses the limitations. This chapter is instrumental in reflecting on the effectiveness of the SPMM and suggesting areas for future research.
- Chapter 10 concludes the thesis by summarizing the main findings and contributions. This chapter also outlines future work, paving the way for ongoing research.

2. Related Work

Contents

This chapter presents several approaches related to the goal of this thesis. First, the work of Sinkovec's master's thesis is discussed, which involved conducting a systematic literature review (SLR) to develop a catalog of security policies in microservice-based applications. [Sin22] Sinkovec's thesis provides a foundational understanding of various aspects of security policies, thereby facilitating further exploration into the automation of these security policies. Next, the chapter examines the work of Rostami et al. which proposes a conceptual model for tailoring information security policies. [RKG22] Finally, we present a collection of publications that discuss security policy automation from various perspectives.

2.1. Security policy aspects for microservice-based applications

This thesis builds upon a growing body of research focused on the intersection of software engineering and security policies in software systems. One foundational work in this area is the master thesis from Sinkovec [Sin22], which provides a comprehensive view on software engineering perspectives of security for microservice-based applications. Sinkovec's master thesis highlights the need for a security meta model tailored towards the unique challenges posed by microservice architectures, emphasizing the importance of various security views and concerns specific to this architectural style.

The concept of a security meta model, as explored by Sinkovec, is particularly relevant to this thesis. It offers a structured approach to understand and integrate security policies within the software engineering life cycle. This aligns with the objectives of our research, which introduce a novel Security Policy Meta Model that aides the automation of Security Policies in software systems.

In addition to the aforementioned insights, Sinkovec's work introduces an extensive "Catalog of Security Design Concepts for Microservices". This catalog is a critical resource for understanding and implementing security in microservice architectures.

In addition to providing a foundational framework for understanding security in microservice architectures, Sinkovec's "Catalog of Security Design Concepts for Microservices" plays a pivotal role in the development and evaluation of the security policy

meta-model introduced in this thesis.

Furthermore, the use of Sinkovec's catalog as an evaluation framework allows for a rigorous assessment of how well the security policy meta-model aligns with established security concepts and practices. This alignment is critical for ensuring that the security policy meta model is not only theoretically sound but also practically applicable in real-world scenarios. The catalog's comprehensive nature aids in identifying any potential gaps in the security policy meta model, ensuring that it adequately captures the diverse and complex nature of security in software systems.

2.2. Conceptual models for tailoring security policies

The necessity of safeguarding critical information in business processes is predominantly addressed through Information Security Policies (ISP). These policies, however, are often not adequately tailored to different target groups within organizations. In response to this, Rostami et al. [RKG22] propose a conceptual model for software that supports the modularization and tailoring of ISPs. Employing design science research, they developed this model as a Unified Modeling Language (UML) class diagram based on ISPs from public agencies in Sweden. This conceptual model aims to serve as a foundation for software that facilitates the customization of ISPs to various stakeholders within an organization.

The research in the realm of software aiding ISP design is not extensive, but it does provide insights into few aspects of security policy tailoring. Prior works have primarily focused on the functionality of security policy without delving into the underlying conceptual models. For instance, one study introduced a framework for evaluating ISP performance [Sya10], yet it did not support the modularization of ISPs. Other researchers have expanded on the information security management toolbox, which follows a process-oriented model [CS13]. This toolbox, though providing a personalized and tailormade document, lacks clarity on how it caters to specific work situations of employees, largely because the controls are derived from the international information security standard ISO 27002 [22a; RKG22].

Additionally, existing research offers several models and frameworks to support ISP design, but they often do not constitute conceptual models in the truest sense and fail to address the nuances of tailoring. An important contribution in this field has been the identification of requirements for software that aids in ISP design, with two primary focuses: supporting a tailorable design of ISPs and addressing clear and uniform target groups. Despite the valuable nature of these requirements as prerequisites for developing software that supports ISP tailoring, they have not yet been transformed into a concrete model or software. [RKG22]

2.3. Approaches for automation of security policies

The current landscape of security policy automation is characterized by several notable approaches. One methodology involves the use of traffic flow modeling, as described by

Bussa et. al. [BSV22]. This approach focuses on network traffic management for network security automation, introducing models such as Atomic Flows and Maximal Flows for traffic representation. However, this approach focuses primarily on the network layer, leaving room for more comprehensive solutions that integrate other system components.

In addition, Rouland et. al. [Rou+23] proposed a methodology for designing secure software architectures. This approach uses formal methods to ensure alignment between security policies and goals. It provides a structured process for verifying and applying security solutions. However, its primary limitation is its focus on component-based architectures, which could overlook other aspects of security policies in software systems.

In addition, Navarro-Machuca et. al. [NC16] present a model-based approach for incorporating security policies directly into the software development life cycle. This strategy allows security policies to be mapped to data fields in business models, thereby facilitating the application of security controls during development. This approach emphasizes the need for collaborative input from a variety of stakeholders involved in the SDLC. While innovative, this model-based strategy primarily addresses security at the data field level, potentially overlooking broader aspects of security policy in software systems.

Another approach, introduced by Horcas et. al. [HPF14], uses Software Product Lines (SPLs) and Aspect-Oriented Programming (AOP). This approach links security policies to functionality and automates the deployment of that functionality. It effectively bridges the gap between theoretical policy specifications and their practical application in software systems. However, while it provides a structured method for translating high-level security policies into implementable software components, it may not fully address the dynamic nature of security policies in software systems.

In addition, Hammar et. al. [HS23] introduce an innovative concept of using high-fidelity emulation systems to create digital twins of IT infrastructures. This approach supports the development and automation of security policies by testing and refining them in a simulated environment prior to implementation. While this approach improves the effectiveness and adaptability of security policies, it focuses primarily on the emulation aspect and may not encompass the full lifecycle of security policy development and enforcement in software systems.

Furthermore, Ranise et. al [RS17] discusses an approach for verifying the compliance of security policies with regulatory requirements. The approach emphasizes static analysis techniques and tools to ensure compliance by design. However, its focus is on legal compliance rather than the technical aspects of policy automation and integration into software systems.

In addition, Abassi et. al. [AE08b] presents an automated tool for validating security policies. This paper presents a three-step validation process, including consistency, completeness, and preservation of security and liveness properties. While this approach is crucial for establishing robust validation frameworks for security policies, it focuses mainly on firewall contexts and may not encompass the broader aspects of security policy automation in other software systems.

Finally, Hamdi et. al [HBM07] explores a novel architecture for automating the

enforcement of security policies in distributed systems. This work contributes to the understanding of how security policies can be embedded and enforced in distributed architectures. However, its specific focus on distributed systems may limit its applicability to other types of software systems or architectures where different approaches to security policy enforcement are required.

Identifying these gaps, this thesis introduces the Security Policy Meta Model (SPMM), offering a novel approach to the automation of security policies in software systems. Unlike existing methodologies, the SPMM provide a framework that integrates security policy considerations from all stakeholders and types of security policies. This approach ensures comprehensive coverage, addressing the multi-faceted nature of automating security policies in software systems.

The SPMM is designed to enable the creation of customized Security Policy Models (SPM) that address the concerns of different stakeholders, ensuring a broader and more comprehensive understanding of security requirements. This feature distinguishes it from approaches that focus narrowly on specific aspects such as network traffic or component-based architectures. The AM supports the modeling of security policy automation, ensuring that security policies are not only theoretically sound, but also practically implementable.

This thesis proposes a SPMM that not only encapsulates the strengths of existing approaches, but also fills the gaps they leave. By offering a comprehensive, SPMM that takes into account the multifaceted nature of security policies. The SPMM represents a significant advancement in the field of security policy automation within software systems. This integrated approach ensures that security policies are not only developed in accordance with the diverse needs of stakeholders, but also seamlessly integrated and automated within the software development lifecycle.

3. Foundations

Contents

This chapter introduces the core terminology used in this thesis. The first section presents the term *Security Policy in Software Systems*, including the definition of *Security Policy Model*, which is utilized in subsequent chapters. The second section introduces the term *Automation of Security Policies*, including the definition of *Automation Model*, which is utilized in subsequent chapters.

3.1. Security Policy in Software Systems

In the realm of software systems, a security policy represents an established framework comprised of rules and principles designed to safeguard the integrity, confidentiality, and availability of system components against unauthorized access, use, alteration, destruction, or disclosure [Pis+07b]. Security policies in software may encompass guidelines that delineate accepted information flows, actions permitted within the software systems, and conditions that denote a secure state of the system [10]. In the domain of software system security policy are usually described "in a more or less formal way, within a particular specification language, a notion of information system, suitable in a particular context, and/or the specification of granted actions in this system, and/or the specification of secure states of this system." [Jau10]. Furthermore security policy can be broken down into buildings blocks that address specific concerns in the SDLC [Sin22].

The distinctions among security policies reflect their varying scopes and the particular aspects they address within the SDLC [MRF21]. For example, certain policies might strictly focus on information flow security, stipulating rules and procedures to prevent unauthorized data transmission between different components or layers of a system.

As technology evolves, the granularity of security policies also deepens [Wro+17]. The transition from high-level policies to specific security design elements becomes a critical exercise during the development of software components [NC16]. This translation ensures that security requirements are not only detailed and robust but are also amenable to verification and validation methods, thereby enhancing the overall security posture of the software system [HPF14].

In summary, a security policy in software systems is a foundational element that encapsulates the necessary measures and standards to protect software systems. The

ability to distinguish and refine these policies through the SDLC is essential for achieving a secure and resilient software system.

Security Policy Model

The Security Policy Model is a model that may encompass the entirety of a security policy or only select aspects of it.

3.2. Automation of Security Policies

Automation of security policies is the process of implementing and managing security measures in an automated fashion [Bas+15] [HBM07]. It involves the use of software tools and algorithms to handle repetitive, standardized tasks involved in the enforcement of security policies [Bas+15]. Automation in this context aims to eliminate the need for manual intervention, increasing efficiency and reducing the potential for human error.

There are different types of automation when it comes to security policies, each addressing various aspects of security management [HBM07]. These types broadly include automated authentication mechanisms, where systems can verify user identities without manual input; automated encryption, which secures data by converting it into unreadable formats; and automated network defense systems [YZG17], such as firewalls and intrusion detection systems which can identify and respond to threats autonomously.

Furthermore, automated compliance checks ensure that systems adhere to industry standards and regulations without continuous human oversight, and automated patch management systems can apply necessary software updates to protect against known vulnerabilities.

By utilizing automation, organizations can proactively manage security risks, maintain high standards of data protection, and streamline the overall security infrastructure [Bas+15]. This approach allows for consistent and timely responses to threats, making security operations more robust and effective [LR16].

Automation Model

The Automation Model is a model for automating the aspects of a security policy as modeled in the Security Policy Model.

4. Lightweight Systematic Literature Review

Contents

In this chapter, we describe our methodology for conducting a lightweight SLR, a central technique for assimilating, appraising, and synthesizing all pertinent research related to a specific question or topic area. Although traditionally prevalent in medical and healthcare disciplines, Kitchenham et al. [Kee+07] have adapted guidelines for using this technique in software engineering research. This adaptation facilitates the aggregation and examination of evidence relevant to specific software engineering topics. The characteristic of an SLR is its methodical and transparent nature, which ensures that each phase and interim finding is meticulously documented.

The primary benefits of conducting an SLR include its transparency, reproducibility, and impartiality. Transparency is achieved by providing a thorough account of each step taken, its underlying rationale, and the resulting results. Reproducibility is ensured by documenting a well-defined methodology that allows other researchers to validate and replicate the SLR. Impartiality is maintained by considering all research within the predefined search scope, thereby minimizing selection bias.

The goal of our SLR is to provide a comprehensive view of the existing research that discusses security policies, their structures, and the automation of these security policies. To accomplish this, we will build on the streamlined SLR proposed by Sincovic [Sin22], which in turn is a modification of the approach outlined by Kitchenham et al. [Kee+07]. The following sections detail the adaptations made to Sincovic's model and our further modifications to fit our research context.

4.1. Developed Search Strategy

In this section, we outline the challenges in developing our search strategy and present the resulting strategy. We have identified four key challenges:

- **Database Selection:** There are many databases to choose from, and each has its strengths and weaknesses.
- **Keyword Selection:** Choosing the right keywords is critical to a successful search.
- **Rapid evolution:** Security policies and their implementation in software systems are subject to constant change and evolution.
- **Proprietary information:** Many companies have their own internal security policies and procedures that are not publicly available due to their sensitive nature. This can limit the amount of practical, real-world information available for review.

To address the first challenge, we present our selection of scientific databases, using Sincovic's master's thesis as a guide. To address the second challenge, we have developed a metasearch query that is applicable to all of the selected databases. The third and fourth challenges acknowledge the inherent limitations of our approach: the rapidly evolving nature of the field and the inaccessibility of proprietary information mean that our SLR results serve only as a first point of inquiry. It is critical to recognize that these results may be incomplete or may not reflect the most recent research advances.

4.1.1. Selection of Scientific Databases

In the SLR, the choice of scientific databases critically influences the breadth and quality of the reviewed research. This section details our approach to database selection, which closely follows the methodology outlined in Sincovic's master's thesis.

Sincovic's thesis provides a strategy for database selection that emphasizes a balance between database coverage and manageability of search results. Using this approach, we began by identifying a preliminary list of databases.

Criteria for Database Selection: Our selection criteria were twofold. First, we sought databases with a wide range of scientific publications relevant to our research topic. Second, we sought databases with robust search capabilities, including advanced filtering options, to effectively narrow the search results to the most relevant studies.

Exclusion and Inclusion of Databases: Certain databases were excluded from our search. Databases with limited access or inadequate filtering capabilities were deemed inappropriate for our needs. Similarly, databases that yielded excessively high search results, thereby complicating the review process, were also omitted.

The final selection of databases was a balanced mix of comprehensiveness and practical considerations of search manageability. We included well-known databases such as the

ACM Digital Library, IEEE Xplore, Web of Science, Science Direct , and Scopus, which are known for their extensive coverage of software engineering literature. These databases not only offered a wealth of relevant publications, but also provided the necessary tools to effectively filter and manage the search results.

4.1.2. Search Query construction process

Guided by our research question, we identified three essential keywords to include in the search query: "security policy", "software system", and "policy lifecycle phase". Using these keywords, we developed the first metasearch query, as shown in Listing 4.1.

```
(security OR secure)
AND
("software system" OR lifecycle OR element OR aspect OR feature OR
    factor)
```

Source Code 4.1: First meta search query

This query, when run against the Web of Science database, returned over 200,000 entries. Due to the impracticality of reviewing such a large number of results, we recognized the need to refine the query. Our goal is to identify papers that focus on the nature of security policies in software systems. To accomplish this, we removed the overly broad keyword "policy lifecycle phase" which could attract irrelevant papers. This modification led to the creation of a revised meta query, which is presented in Listing 4.2.

```
("security polic*")
AND
(model OR component OR propert*)
AND
software
```

Source Code 4.2: Revised meta search query

The revised search returned 658 entries from various databases. A preliminary review of these results revealed that many of them did not extensively discuss security policy in software systems. This insight led to further refinement of the query. The final version, detailed in Listing 4.3, narrows the scope to entries with "security policy" in the title and the term "software" in the abstract.

```
TITLE:(security AND polic*)
AND
ALL:(model* OR component* OR propert*)
AND
ABSTRACT:(software)
```

Source Code 4.3: Final meta search query

Implementing this final query across all selected databases returned 326 results. In the appendix, the Table A.1 provides an overview of the results, grouped by database, and the specific query used for each database.

4.2. Study Selection

This section outlines the selection process for the results of the meta-query execution. First, the inclusion and exclusion criteria are presented. These criteria are then applied to the search results. The first phase focuses on the evaluation of the title, abstract and conclusion of the publications. In the second phase, the focus shifts to a comprehensive analysis of the full publications.

Inclusion and Exclusion criteria

In this section, we present the inclusion and exclusion criteria for our study selection process. The exclusion criteria are designed to filter out publications that are not suitable for inclusion for technical reasons. For example, we exclude publications that are duplicate or not written in English. The defined exclusion criteria are:

Exclusion Criteria:

- **Duplication:** We exclude publications that are duplicates, appearing in multiple database query results.
- **Not in English:** Only publications in English are considered to ensure full understanding and thorough analysis.
- **Availability of full text:** We exclude publications where the full text is not available.
- **Lack of peer review:** Publications that have not been peer reviewed are excluded.

Inclusion Criteria:

- **Relevance to security policy in software systems:** We include papers that specifically discuss security policies in software systems, in line with our primary research focus.
- **Structure and key aspects of security policies:** Papers that address the structure and key elements of security policies are included.

With these criteria established, we applied them to the titles, abstracts, and conclusions of our search results, resulting in an initial reduction of relevant publications to 110.

In the second phase, we extended our evaluation to the full text of these publications. By reapplying the inclusion and exclusion criteria, the number of relevant publications was further reduced to 57.

The final list of relevant papers is included in the Appendix of this thesis. A

4.3. Data Extraction

A straightforward approach to data extraction is used. Each paper is reviewed to extract information relevant to the three research questions. For each research question, detailed notes are compiled highlighting key information from the papers that contributes to addressing the research questions. This is followed by a summary of the extracted information and identification of potential gaps in existing research. The refined notes, corresponding to each paper and research question, are available in the appendix of this master's thesis. B

4.4. Results

In this section, the results of the conducted SLR are presented and the research gap in relation to our research questions is presented. In the first subsection, the results regarding the key aspects and structure of security policies are presented. In the second subsection, the possibilities for tailoring security policies are discussed. In the last subsection, the challenges of automation in security policies are presented.

4.4.1. Key aspects and structure of Security Policies

Security policies in software systems are multifaceted and diverse, each tailored to specific requirements and environments. [XM03] The structure of these policies varies significantly across different systems. For instance, event flow security policies emphasize controlling the propagation of event flows in the system, ensuring that sensitive data is accessible only to authorized entities. This approach detaches the security policies from the functional implementations of processing units, focusing instead on end-to-end guarantees for data security. [Mig+10]

In addition, Access control mechanisms, another aspect of security policies, typically consist of rules specifying permitted or denied access to various resources under certain conditions. Policy specification languages like XACML facilitate defining these access rights with precision and flexibility. [SY13] [HAC19]

Furthermore, Norms, including commitments, authorizations, and prohibitions, play a significant role in structuring security policies. [Kaf+17] They formalize expected user behaviors and establish accountability frameworks, crucial for distinguishing between accidental and malicious security breaches. [LR16]

A notable gap in the literature is the detailed exploration of the precise structure and content of security policies. While several publications address the assessment and testing of security policy changes, more thorough insights into the structural details of these policies and their practical applications in various software environments are needed.

In summary, security policies in software systems are characterized by their emphasis on data confidentiality and integrity, complex structural layers, and adaptability to evolving environments. Despite substantial progress in policy design and implementation, the field still requires deeper research, particularly in understanding the intricate structures and practical implications of these policies in diverse software settings.

4.4.2. Tailoring Security Policies for the Concerns of Stakeholders

The key findings on the research question (RQ2) "How can the aspects of security policies in software systems be tailored towards the concerns of stakeholders?" can be summarized as follows:

- **Security Policy Tailoring Approaches:** The reviewed papers primarily focuses on adopting and adjusting existing security policies rather than modeling them for varied stakeholder understanding. This includes developing high-level security policies abstracted from specific implementations, parameterization of security policies, defining flow categories, and establishing event flow constraints. [Mig+10; BBR15; Hag+18; H19]
- **Stakeholder-Specific Requirements:** Tailoring involves understanding each stakeholder's specific security requirements and incorporating them into the design of security policies. This may include considering the diverse needs of health institutions, financial organizations, and other entities. [Mig+10; BBR15; H19]
- **Flexibility and Customization:** Some publications advocate for a policy-based security model adaptable to user-specific security needs. This model includes various enforcement methods, monitoring and response mechanisms, and lifecycle management. [Hag+18; H19]
- **Lack of Comprehensive Methodologies:** While some publications offer use case scenarios and flexible architecture for implementing security policies, they do not provide a comprehensive framework or methodology for engaging with stakeholders to ascertain their specific security concerns.

In conclusion, while the existing papers offers various approaches to tailor security policies in software systems, it predominantly emphasizes technical adjustments and implementations. There is a noticeable gap in the direct modeling of security policies for diverse stakeholder comprehension and engagement, indicating an area for further research and development.

4.4.3. Possibilities and challenges of automating security policies

The research presented in the extracted data from the SLR focuses primarily on the technical aspects of automating security policies in software systems. This focus is evident in the various methods and challenges discussed, which focus primarily on technical solutions and difficulties.

For example, discussions revolve around event-driven policy enforcement, information flow control [Mig+10], static analysis models for RBAC systems [Pis+07a], and integrating access control requirements into software development processes [SY13]. These aspects are highly technical, dealing with system architecture, coding practices, and algorithmic implementation.

In addition, specific contexts such as SoC design [BBR15; Hag+18; MRF21], middleware design [Mig+10], and SDN-based cloud environments [Wro+17; Soo+19; YZG17; LR16; Pis+19; H19] underscore this technical orientation.The focus is on aspects such as reducing complexity, improving validation and debugging, dynamic configuration, and automatic conflict resolution - all of which are technical in nature.

However, this focus overlooks the comprehensive involvement of all stakeholders in the SDLC. Stakeholders include not only developers and technical managers, but also end-users, business analysts, compliance officers, and other non-technical participants. These stakeholders have different concerns and perspectives that extend beyond the technical realm.

The omission of these broader perspectives can be seen in the lack of discussion of how automated security policies might affect user experience. There's little mention of stakeholder involvement in the definition or evaluation of these security policy automation, or in the decision-making processes that guide the development and implementation of such automated systems.

This technical-centric approach can lead to gaps in understanding the full impact of security policy automation on various aspects of the SDLC and the operation of the system in real-world scenarios. It can result in solutions that, while technically sound, may not be well aligned with the needs and expectations of all stakeholders involved in or affected by the software system.

In conclusion current research, as represented in the SLR, shows a clear bias toward the technical aspects of automating security policies in software systems, lacking a holistic view that incorporates the diverse range of stakeholders involved in the SDLC.

4.5. Excluded SLR steps

In the previous section, we mentioned that we are conducting a lightweight SLR based on the modified SLR approach proposed by Sincovic. The primary goal of our SLR is to improve our fundamental understanding of security policy automation and the general principles that govern it. The main objective is to develop a meta-model and conceptual models to bridge the gap in modeling security policy automation, tailored to address the concerns of all stakeholders in the Software Development Life Cycle (SDLC) of software systems. Consequently, the additional steps suggested by Kitchenham et al. are not implemented in this thesis, as this would be out of scope.

5. Security Policy Concept Model

Contents

This chapter introduces the SPCM and the SPMM. The SPCM is designed to simplify the process of constructing a model for a security policy that is tailored to stakeholder concerns. The SPMM facilitates the integration of the SPCM with the Security Policy Automation Concept Model introduced in the following Chapter 6.

The chapter is divided into three sections: model elements, their relationships within the model, and the integration into the SPMM. The first section provides a fundamental understanding of the model elements. The following section examines the relationships between these elements. Finally, this study explores the integration of elements into the SPMM.

To improve clarity and comprehensibility, we present a visualization of the SPCM in Figure 5.1. This serves as a visual aid to navigate the concepts discussed in this chapter.

5.1. Model Elements

This section explores the fundamental elements of our SPCM: *Stakeholder*, *Security Policy*, *Concern*, *Security Policy Perspective*, *View*, *Model Kind*, and *Viewpoint*. Although elements such as *Stakeholder*, *Concern*, *View*, *Model Kind*, and *Viewpoint* are based on the principles of existing software architecture metamodels [NIC08].

Each subsection provides an overView of the elements of the SPCM. Firstly, each element is defined precisely to establish a clear understanding of its fundamental concept. Following this, the purpose of each element's existence in the SPCM is detailed. Finally, practical examples are provided for each element. These examples demonstrate the application of the theoretical concepts in real-world contexts.

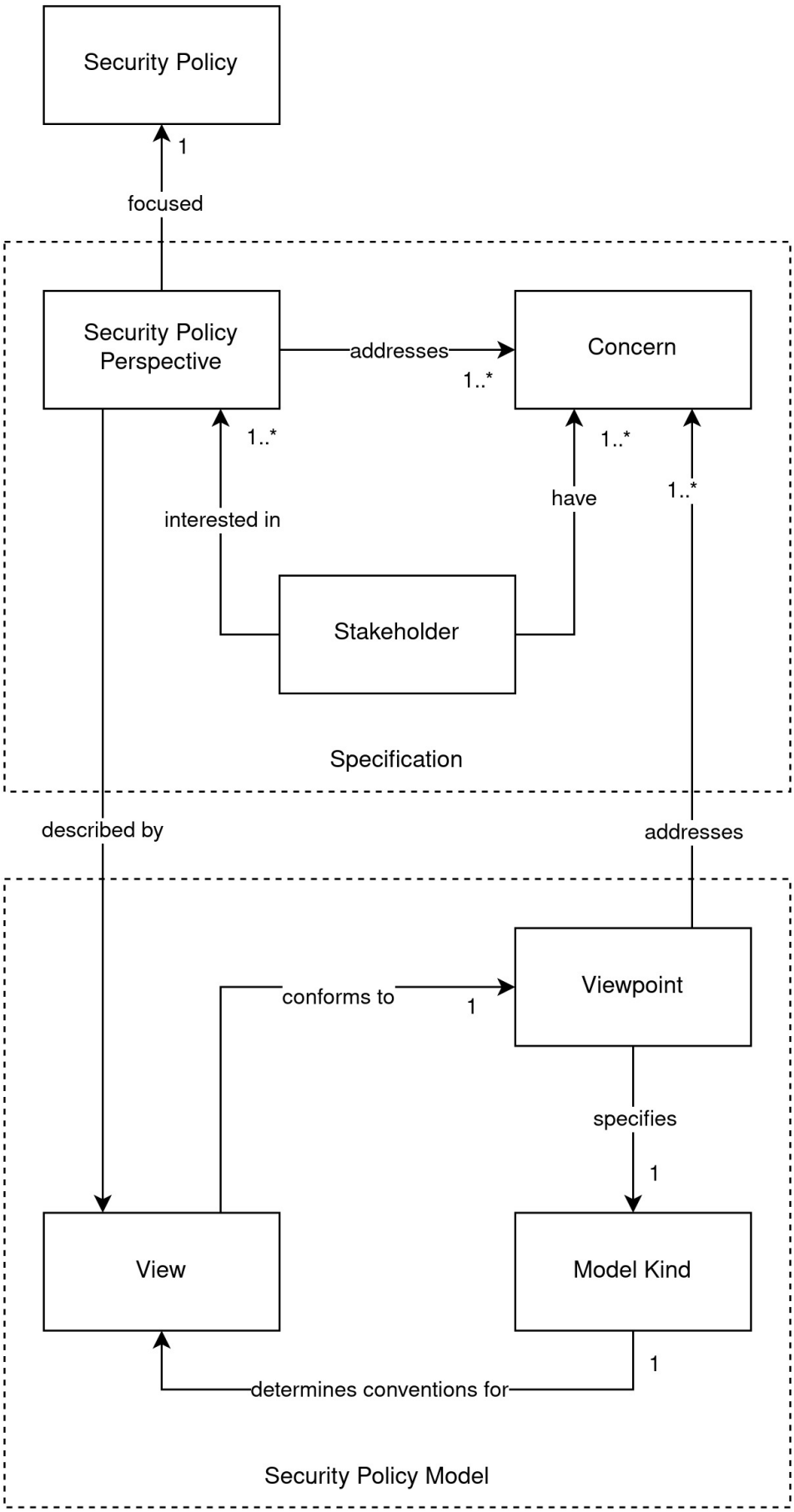

Figure 5.1.: Security Policy Concept Model

5.1.1. Security Policy

Definition: Security Policy

A *Security Policy* is a documented set of policies, procedures, or constraints that outlines an organization's requirements for maintaining a specified level of security. The *Security Policy* establishes the foundation for identifying, assessing, and mitigating security risks throughout the Software Development Life Cycle. The objective of a *Security Policy* is to ensure that software is developed to maintain the confidentiality, integrity, and availability of information.

The primary objective of this element is to incorporate the target security policy into the resulting model. This element guides the resulting model, ensuring that all other elements align with the target security policy.

Examples

Least Privilege Policy: This *Security Policy* mandates that every system entity, such as a user, process, or program, should only have access to the resources and information essential for its intended function. For instance, in a web application, only authenticated users with a 'Viewer' role can access an API endpoint that retrieves user profiles, while unauthenticated users or authenticated users with a different role cannot. [Sin22]

Strong Authentication Policy: This *Security Policy* mandates the implementation of strong authentication mechanisms, which often require multiple verification factors to confirm the identity of users or entities. As an example, a cloud service that enforces this *Security Policy* may require users to provide a combination of a password and a one-time token generated on a mobile device for access. This ensures a stringent two-factor authentication process. [Sin22]

Data Encryption Policy: This *Security Policy* emphasizes encrypting sensitive data, both at rest and in transit, to prevent unauthorized access and ensure confidentiality. For instance, an application that manages user credentials should use AES-256 encryption to protect data at rest in its database. Additionally, the application should deploy TLS 1.3 encryption for data in transit to ensure secure communication. [Sin22]

5.1.2. Stakeholder

Definition: Stakeholder

A *Stakeholder* is an individual, group or organization that has an interest or concern in the compliance and implementation of the *Security Policy*. *Stakeholders* may have active involvement in the project as software developers, end-users, or project managers, or passive involvement as customers, regulators, or the general public. [21]

This element is the central figure in the resulting model. All elements of the SPCM, except for the *Security Policy* itself, are defined directly or indirectly in relation to it.

Examples

Software Developers: Software developers are primarily concerned with integrating security protocols into software design and development processes.

System Administrators: System Administrators are key contributors in managing and maintaining an organization's IT infrastructure. Their duties include enforcing security policies and addressing security breaches.

5.1.3. Concern

Definition: Concern

A *Concern* is an interest, objective, necessity, or issue related to the *Security Policy* that is important to a *Stakeholder*. [18]

A *Concern* must be significant to the *Stakeholders* involved, meaning it should directly impact their interaction with or expectations of the *Security Policy*. In addition, a *Concern* must be relevant to the *Security Policy* and its *Stakeholders*. Identifying and describing *Concerns* within the SPCM is critical to ensuring that the resulting model is comprehensive and meets *Stakeholder* requirements.

Examples

Compliance with Regulations and Standards: Compliance with industry regulations, such as GDPR in Europe, HIPAA in healthcare, or PCI DSS in payment processing, is crucial to avoid legal and financial consequences.

Training and Awareness: Ensuring that all members of an organization are aware of and adhere to security policies.

Incident Response and Recovery: In the event of a security breach, it is important to respond and recover with speed and efficiency.

5.1.4. Security Policy Perspective

Definition: Security Policy Perspective

A *Security Policy Perspective* is a detailed and stakeholder-focused perspective of the overarching *Security Policy*. As a filter, this perspective emphasizes and expands upon those aspects of the *Security Policy* that are most relevant to specific *Stakeholder Concerns*.

The purpose of this element is to simplify the *Security Policy* into a form that individual *Stakeholders* can easily understand and find relevant, ensuring that the most important *Concerns* of a given *Stakeholder* are addressed. This approach leads to a *Security Policy* that is focused and easier for *Stakeholders* to understand. The main objective of this element is to ensure that resulting models are created with clarity and direction, and that they align with the *Concerns* of *Stakeholders*.

Examples

System Administrators Perspective of a Data Encryption Policy: System Administrators 5.1.2 are tasked with implementing a *Data Encryption Policy* 5.1.1 by securing data at rest with robust encryption such as AES-256 and ensuring that data in transit is protected using TLS 1.3 protocols. Their role is essential in configuring, managing, and maintaining encryption standards throughout the organization's IT infrastructure, including handling encryption keys and verifying the efficacy of the security policy.

Software Developer Perspective of Data Encryption Policy: The focus of the software developer 5.1.2 is to incorporate secure encryption methods into the software's architecture. This includes integrating AES-256 encryption to safeguard data stored within databases and ensuring that the application employs TLS 1.3 for secure data transmission. The role of a Software Developer also involves understanding and applying best practices in encryption and ensuring that the software they develop complies with the latest security standards. They are responsible for writing code that efficiently implements encryption technologies while maintaining system performance.

5.1.5. Viewpoint

Definition: Viewpoint

A *Viewpoint* is a guideline for creating, interpreting, presenting, and analyzing the View. [Per19; 19] The *Viewpoint* outlines the information and criteria necessary to evaluate whether the *View* addresses the *Stakeholder Concerns*.

This element is crucial for establishing the *View* and *Model Kind*. It serves as the basis for interpreting and constructing the *View*, ensuring that all *Concerns* are adequately addressed and an appropriate *Model Kind* is defined for the effective creation of *Views*.

Examples

Corporate Data Protection: This *Viewpoint* focuses on the importance of protecting sensitive data in the corporate sector. The formation of *Security Policies* is dictated by the unique *Concerns* of corporate *Stakeholders*, such as the risk of data breaches and regulatory compliance.

5.1.6. Model Kind

Definition: Model Kind

A *Model Kind* is a semantic and syntactic specification that outlines the construction of a *View*. It defines the building blocks, modeling language, and structural constraints that a *View* can use. [22b]

A *Model Kind* provides semantic and syntactic rules for constructing *Views*. The semantics ensure that the *Views* maintain meaning within the context of *Security Policies*, while the syntax governs the structure and form of the *View*. In addition, it serves as a guide to ensure that all necessary and relevant aspects of a *Security Policy* can be represented, and that there is consistency in how these aspects are modeled in the *View*. The purpose of a *Model Kind* is to ensure that *Views* are created consistently, comprehensively, and understandably within the context of *Security Policies*.

Examples

The following paragraphs provide conceptual examples, while the Chapter 8 contains concrete examples.

Data Modeling: The *Data Modeling Model Kind* defines the methodologies for representing data elements within software processes. It requires the use of specific modeling languages, such as Entity-Relationship (ER) diagrams or *Unified Modeling Language (UML)*, and establishes strict guidelines for depicting data relationships and flows.

Workflow Process: The *Model Kind* defines the standards for representing workflow processes in a system. It offers a comprehensive framework for illustrating the different stages of a process, including decision points and activity flows. The use of standardized notation, such as *Business Process Model and Notation (BPMN)*, is required to ensure precise and consistent representation of complex workflow processes in the *Views*.

5.1.7. View

Definition: View

A *View* is a concrete model of a particular aspect of the *Security Policy*. It is created based on a set of syntactic and semantic rules established by its corresponding *Model Kind*, guaranteeing that the *View* is internally consistent and in line with the overall goals of the *Security Policy*. [22b; NIC08]

This element is the concrete model utilized by *Stakeholders* to improve their understanding of *Security Policies* or to implement them. Hence, the element is a vital component in the resulting model.

Examples

The following paragraphs provide conceptual examples, while the Chapter 8 contains concrete examples.

Database Administrator's View: This *View* conforms to the *Data Modeling Model Kind* 5.1.6 and presents a detailed model of the *Security Policy* with a focus on data management and storage protocols. The *View* includes comprehensive specifications for data encryption standards, access control mechanisms, and data retention policies.

Process Manager's View: Adhering to the *Workflow Process Model Kind* 5.1.6, this *View* illustrates the integration of security measures within business process workflows. It demonstrates the inclusion of security checkpoints in business processes, the transfer of sensitive information between organizational departments, and the procedures for handling security incidents within these workflows.

5.2. Model Relations

This section presents the model relationships and explains how the elements of the SPCM interact.

The relationships of the SPCM element are unidirectional and identified by distinct names. By default, unless otherwise specified, the relationship's cardinality is set to zero-to-many.

Security Policy Perspective to Security Policy: A *Security Policy Perspective* **focus** one *Security Policy*. With this focusing the perspective of the *Security Policy* is tailored towards a *Stakeholders* and there *Concerns*.

Stakeholder to Concern: A *Stakeholder* **has** one to many *Concerns*. This design choice is intended to reflect real-world scenarios where a *Stakeholder* typically has several

different *Concerns*. This relationship does not restrict multiple *Stakeholders* from having the same *Concerns*.

Stakeholder to Security Policy Perspective: A *Stakeholder* is **interested in** one to many *Security Policy Perspectives*. It can be assumed that a *Stakeholder* is not only interested in a *Security Policy Perspective* explicitly tailored to its specific *Concerns*, but also in the *Security Policy Perspectives* of other *Stakeholders*. In addition, it may be beneficial to define multiple *Security Policy Perspectives* for the same *Stakeholder* with disjunctive or different subsets of the *Stakeholder*'s *Concerns*.

Security Policy Perspective to Concern: A *Security Policy Perspective* **addresses** many *Concerns*. Similar to the relationship between *Stakeholder* and *Concern*, and *Stakeholder* and *Security Policy Perspective*. Multiple *Security Policy Perspectives* can address a single *Concern*. By considering these three relationships, it is possible to select the addressed *Concern* to define a *Security Policy Perspective* that directly addresses multiple *Stakeholders*.

Viewpoint to Concern: A *Viewpoint* **addresses** one to many *Concerns*. Similar to the relationship between a *Security Policy Perspective* and a *Concern*. By carefully selecting the *Concern* addressed, *Viewpoints* can be defined that address the *Concern* of multiple *Stakeholders*.

Viewpoint to Model Kind: A *Viewpoint* **specifies** one or many *Model Kinds*. Because the *Model Kind* reflects the *Viewpoint* specification, a *Model Kind* can be associated with only one *Viewpoint*.

Model Kind to View: A *Model Kind* **determines conventions for** one or many *Views*. This relationship is similar to the relationship between *Viewpoint* and *Model Kind*. To ensure *View* cohesion, a *View* can only be associated with a single *Model Kind*.

View to Viewpoint: A *View* **conforms to** one *Viewpoint*. Similar to the relationship between *Model Kind* and *View*, a *Viewpoint* can be associated with multiple *Views*.

Security Policy Perspective to View: A *Security Policy Perspective* is **described by** one or many *Views*. Again, since a *View* is limited to exactly one *Viewpoint*, it would be possible to select the concerns that a *Security Policy Perspective* and *Viewpoint* address in a way that tailors the *View* to multiple or single *Stakeholders*.

5.3. Security Policy Meta Model

With the presented SPCM we can on define a SPMM 5.2. To do this we can group the relationships and elements into two main relationship clusters.

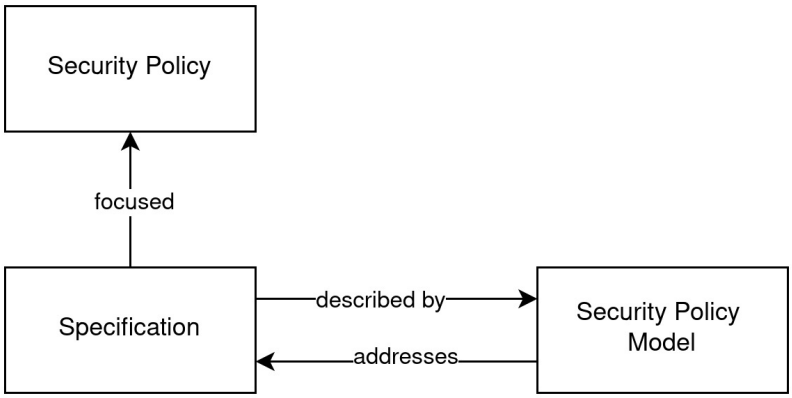

Figure 5.2.: Partial Security Policy Meta Model

- **Specification:** This is the interaction between the *Security Policy Perspective*, *Stakeholder* and *Concern*. The *Specification* is a customized specialization of the *Security Policy*.
- **Security Policy Model:** This involves the interactions between the *View*, *Viewpoint*, and *Model Kind*. with the resulting SPM being the model of the *Security Policy*.

With this SPMM, the separation into two abstract elements, *Specification* and SPM, is achieved.

In the Chapter 6, the SPMM is extended by incorporating the AM element. This integration facilitates the automation of *Security Policies* within our SPMM.

6. Security Policy Automation Concept Model

Contents

In this chapter, we expand the SPMM to include a Concept Model dedicated to automating SPMs. This automation focuses on implementing the concepts modeled within the SPM, rather than on generating the models themselves. For instance, a SPM that outlines responses to *Security Policy* breaches could be translated into a script that executes these defined steps. Alternatively, a model depicting system architecture might facilitate automation by generating parts of the necessary source code or configurations. It's important to note that the output of this automation might not always be executable code or configurations; in some cases, it may be textual instructions. These instructions guide *Stakeholders*, for whom the SPM is constructed, in manually implementing the modeled *Security Policy*. The Chapter 7 will go deeper into this aspect.

This chapter is organized into three sections: the presentation of model elements, their relationships within the model, and the integration into our SPMM. The first section provides a fundamental understanding of the model elements. Next, we examine the relationships between these elements. Finally, we investigate the element integration into the SPMM.

To enhance clarity and provide a comprehensive outlook, we present a diagram of the SPACM as shown in Figure 6.1, serving as a visual aid to navigate the discussed concepts in this chapter.

6.1. Model Elements

In this section, we explore the elements of the SPACM: *Automation Strategy*, *Automation Solution*, *Concern*, *View*, and *Model Kind*. The elements *Concern*, *View*, and *Model Kind* elements are the same as the elements from our SPCM 5.

Each subsection provides an overview of the elements of the SPACM. Firstly, each element is defined precisely to establish a clear understanding of its fundamental concept.

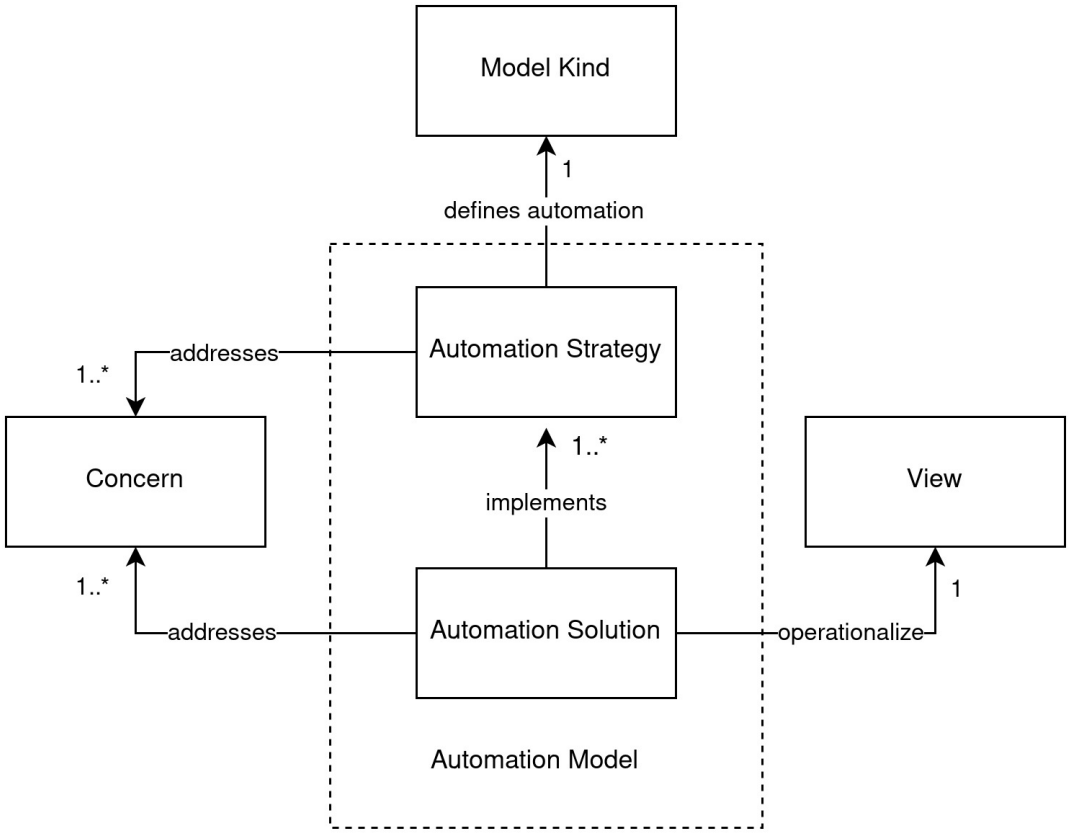

Figure 6.1.: Security Policy Automation Concept Model

Figure 6.2.: Example for concrete View

Following this, the purpose of each element's existence in the SPACM is detailed. Finally, practical examples are provided for each element. These examples demonstrate the application of the theoretical concepts in real-world contexts.

6.1.1. Automation Strategy

Definition: Automation Strategy

An *Automation Strategy* is a detailed description of an automation concept for each syntactic and semantic concept of a *Model Kind*.

The *Automation Strategy* is defined by a systematic approach that includes the identification, design, and application of automatable actions that are specifically tailored to the syntactic and semantic aspects of the *Model Kind*. Furthermore, the integration of an *Automation Strategy* within our SPACM serves the purpose of providing a clear and systematic methodology for automating the modeled *Security Policy*.

Example

This example uses the *View* shown in Figure 6.2. This *View* represents the process of sending a GET request to a URL. Subsequently, the *Model Kind* is restricted to the syntax and semantics shown in the *View*. The *Model Kind* is characterized as a BPMN using only the start and end events and a predefined task, specifically the BPMN task "send a GET request to a URL".

The *Automation Strategies* that apply to the defined *Model Kind* are preset in the following sections. The *Automation Strategies* depend on the requirements of the *Stakeholders*. The first *Automation Strategy* examples involve creating a bash script to automate the outlined process. The second *Automation Strategy* examples involves a sequence of instructions to execute the objective of the process.

BASH-Script Example: The *Automation Strategy* is to define bash script fragments for each element of the *Model Kind*.

The Listing 6.1 provides an *Automation Strategy* for the BPMN event 'None-Start'. This script fragment indicates that the script should use the bash interpreter.

```
#!/bin/bash
```

Source Code 6.1: Automation Strategy for BPMN-Event None-Start

The Listing 6.2 provides an *Automation Strategy* for the BPMN-Task 'send a GET request to a URL'. This script fragment is designed to send an HTTP GET request to a specified URL.

```
http get \$URL
```

Source Code 6.2: Automation Strategy for BPMN-Task send a GET request to a URL

The Listing 6.3 provides an *Automation Strategy* for the BPMN-Event 'End'. This script fragment performs the function of exiting the script and returning a status code indicating a successful execution.

```
exit 0
```

Source Code 6.3: Automation Strategy for BPMN-Event End

Manuel Actions Example: The *Automation Strategy* is the definition of manual actions that a *Stakeholder* can perform for each element of the *Model Kind*.

BPMN-Event None-Start

Opening the Postman application

BPMN Task "send a GET request to a URL"

Enter the URL with the HTTP method set to Get, and then click the Send Request button.

BPMN End Event

Closing the Postman application

6.1.2. Automation Solution

Definition: Automation Solution

An *Automation Solution* is a composition of *Automation Strategies* that operationalize a *View*.

The *Automation Solution* has key attributes that increase its effectiveness. One such attribute is Integration Capability, which emphasizes the ability to integrate various components of the *Automation Strategy* for a cohesive operationalize of the *View*. Another

key attribute is scalability, which refers to the ability of the solution to efficiently scale in complexity and size to meet the evolving needs of the *View*. The primary purpose of the *Automation Solution* is to implement the theoretical constructs of the *Automation Strategy* into practical operational solutions.

Examples

We will develop an *Automation Solution* based on the automation strategies outlined in the previous subsection 6.1.1 and the *View* presented in Figure 6.2. The first strategy is to automate the process using a bash script. Accordingly, the *Automation Solution* corresponding to the *View* presented earlier (Figure 6.2) is exemplified by the bash script shown in Listing 6.4.

```
#!/bin/bash
http get \$URL
exit 0
```

Source Code 6.4: Automation Solution Example 1

For the second *Automation Solution*, the automation is defined as a list of tasks that the *Stakeholder* has to preform:

1. Opening the Postman applicati
2. Enter the URL with the HTTP method set to Get, and then click the Send Request button.
3. Closing the Postman applicatio

6.2. Model Relations

This section presents the model relationships and explains how the elements of the SPACM interact.

The relationships of the SPACM element are unidirectional and identified by distinct names. By default, unless otherwise specified, the relationship's cardinality is set to zero-to-many.

Automation Strategy to Model Kind: An *Automation Strategy* **defines automation** of one *Model Kind*.

Automation Strategy to Concern: An *Automation Strategy* **addresses** one or many *Concerns*.

Automation Solution to Automation Strategy: An *Automation Solution* **implements** one or many *Automation Strategies*.

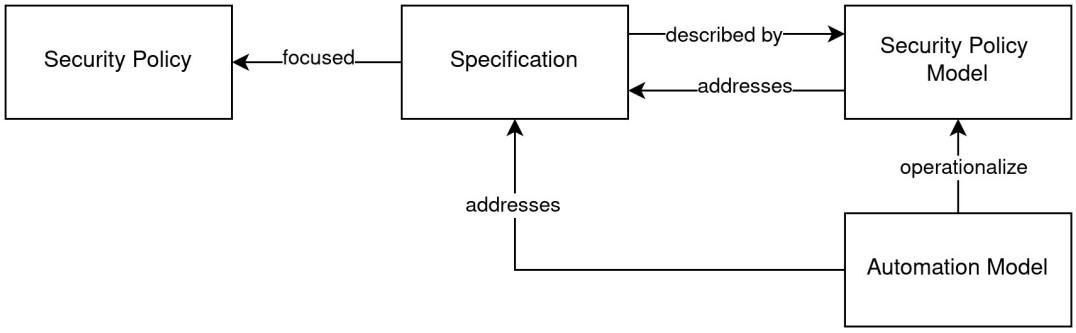

Figure 6.3.: Security Policy Meta Model

Automation Solution to View: An *Automation Solution* **operationalize** one *View*.

Automation Solution to Concern: An *Automation Solution* **addresses** one or many *Concerns*.

6.3. Integration into the Security Policy Meta Model

We can extend the previously introduced SPMM by adding the AM element. This addition is consistent with the existing structure and mirrors the *Specification* and SPM elements. The AM element has two sub-elements: *Automation Strategy* and *Automation Solution*. This extension leads to the creation of the SPMM, which establishes clear relationships between the AM element and both the *Specification* and SPM elements. Within this revised SPMM, the AM element is designed to operationalize the SPM and address specific *Concerns*. Figure 6.3 illustrates this extended SPMM.

7. Security Policy Automation Model Construction Process

Contents

In this chapter we propose the CP for the introduced SPMM 6.3. The goal of the CP is to be able to construct the *Security Policy Automation Model (SPAM)* 7.

Security Policy Automation Model

A SPAM is a model of a *Security Policy*, or aspects of a *Security Policy*, tailored to the *Concerns* of *Stakeholders*. This model includes the *Specification*, the SPM, and the AM.

The first section presents a high-level approach to the process of constructing a SPAM. The second section provides a detailed explanation of each step and its artifacts.

7.1. High level Construction Process of a Security Policy Automation Model

This section presents the high-level process of constructing a SPAM. Figure 7.1 illustrates this construction process and provides a visual representation of these steps:

1. **Specification Construction:** First, we create a tailored *Specification*. This involves considering the *Stakeholders* and their *Concerns*, and defining the *Security Policy Perspective* based on the given *Security Policy*. As a result of this task, the artifact *Specification* is created.

2. **Model Construction:** Next, we construct models for the *Security Policy*. This step involves defining *Viewpoints*, *Model Kinds*, and *Views*. The result of this task is the SPM artifact.

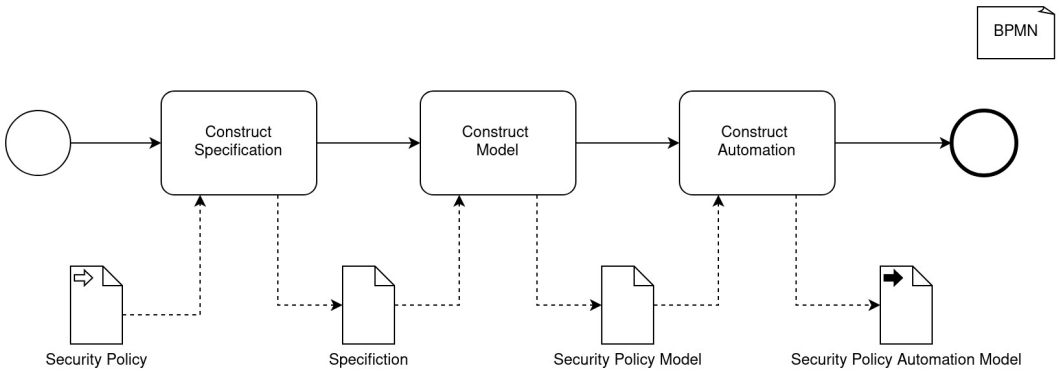

Figure 7.1.: High Level Construction Process of a Security Policy Automation Model

3. **Automation Construction:** Finally, we construct automation for the SPM. This involves constructing *Automation Strategies* for each syntactic and semantic concept of a *Model Kind* and constructing *Automation Solutions* for each *View*. The result of this task is the SPAM artifact.

7.2. Detailed Construction Process of a Security Policy Automation Model

In this section, the CP of a SPAM is presented in detail. Analogous to the High Level Construction Process of a Security Policy AM 7.1, each artifact is presented in a separate subsection. First, the detailed construction process of the *Specification* is presented, then the detailed construction process of the SPM is presented, and finally the detailed construction process of the automation part of the SPAM is presented.

7.2.1. Specification

This section presents the process for constructing the *Specification* artifact. Figure 7.2 illustrates the BPMN process for constructing the artifact *Specification*. In the following section, each BPMN task and its associated artifacts are discussed in detail.

Stakeholder in context of the Security Policy: The first task constructs the *Stakeholder* with the *Security Policy* artifact. In order to accurately reflect the *Stakeholders* in the context of the *Security Policy*, a more focused description of the *Stakeholders* may be required. This is especially important when the *Stakeholders* and the *Security Policy* are from different domains. In the following sections and chapters, this artifact will use the name *Stakeholder*, as this customization is not necessary in most cases.

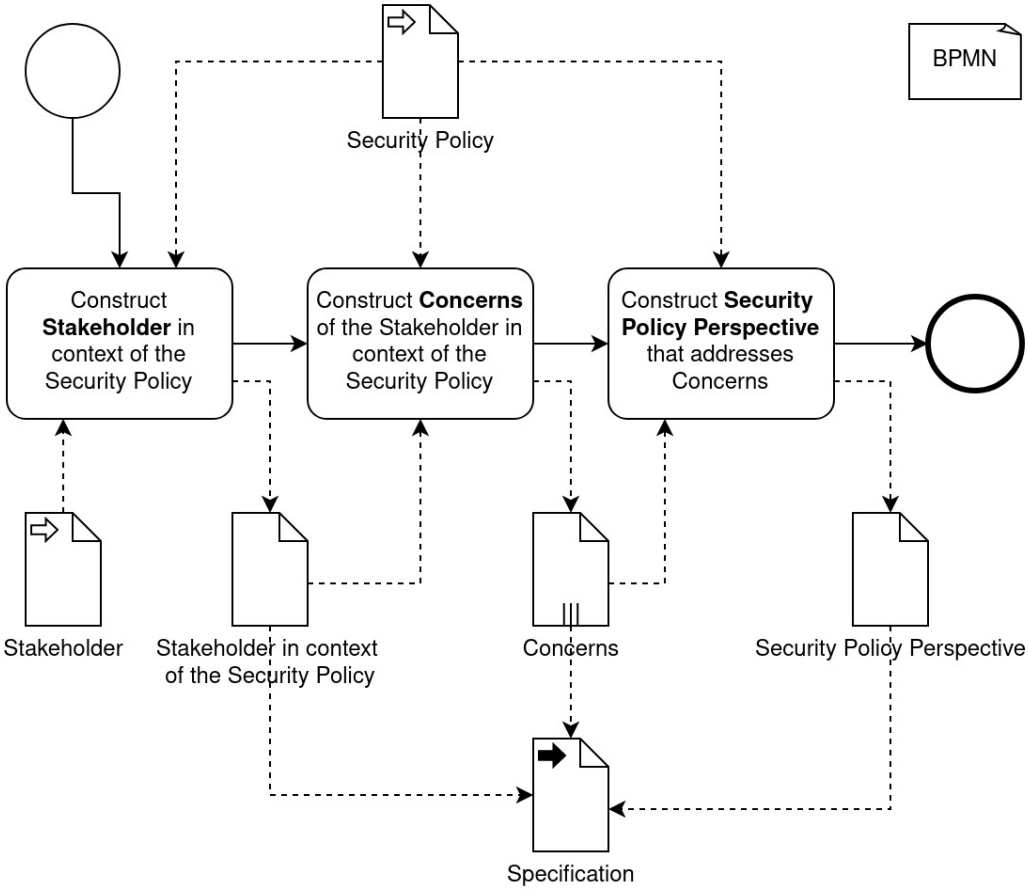

Figure 7.2.: Construction process for the artifact Specification of the Security Policy Meta Model

Concerns: The second task is to construct the *Concern* artifacts. Similar to the construction of the *Stakeholder*, the *Concerns* should not only be the generic *Concerns* of the respective *Stakeholder*, but should be in the context of the *Security Policy*.

Security Policy Perspective: The final task is to construct the *Security Policy Perspective* artifact. To construct a *Security Policy Perspective*, it is necessary to gather artifacts from previous steps, including *Stakeholders* and *Concerns*, as well as the *Security Policy* itself. As described in Section 5.1.4, the *Security Policy Perspective* should address the selected *Stakeholder Concerns*. It is important to note, as mentioned in Section 5.1.4, that not all of a *Stakeholder*'s *Concerns* need to be addressed. By selecting a subset of *Concerns*, it is possible to construct a *Security Policy Perspective* that effectively addresses the *Concerns* of multiple *Stakeholders* 9.1.1.

Specification: With the *Stakeholder*, *Concerns* and *Security Policy Perspective* of the artifacts, the artifact *Specification* is constructed and can be used in the next task of the high-level construction process of a SPAM 7.1.

7.2.2. SPM

This section presents the process for constructing the SPM artifact. Figure 7.3 illustrates the BPMN process for constructing the SPM artifact. In the following section, the individual BPMN tasks and their associated artifacts are discussed in detail.

Viewpoint: The first task constructs the *Viewpoint* artifact. To construct a *Viewpoint*, only the *Concern* artifacts are strictly necessary. While respecting the *Security Policy* and *Stakeholders* is important, the primary focus should be on the *Concerns* to facilitate the creation of a *Viewpoint* tailored to multiple *Stakeholders*.

Model Kind: The second task constructs the *Model Kind*. To construct a *Model Kind*, the artifact *Viewpoint* is essential. This *Viewpoint* provides a *Specification* for interpreting and constructing the resulting *View*. Defining the *Model Kind* allows us to specify the syntax and semantics of *Views* that conform to the established *Viewpoint*.

View: The tired and final task is to construct the *View* artifact. With the constructed *Viewpoint* and *Model Kind*, we can now construct a *View* that models the *Security Policy*.

SPM: With the artifacts *Viewpoint*, *Model Kind*, and *View*, the output artifact SPM is constructed and can be used in the next task of the High-Level Construction Process of a Security Policy Automation Model.

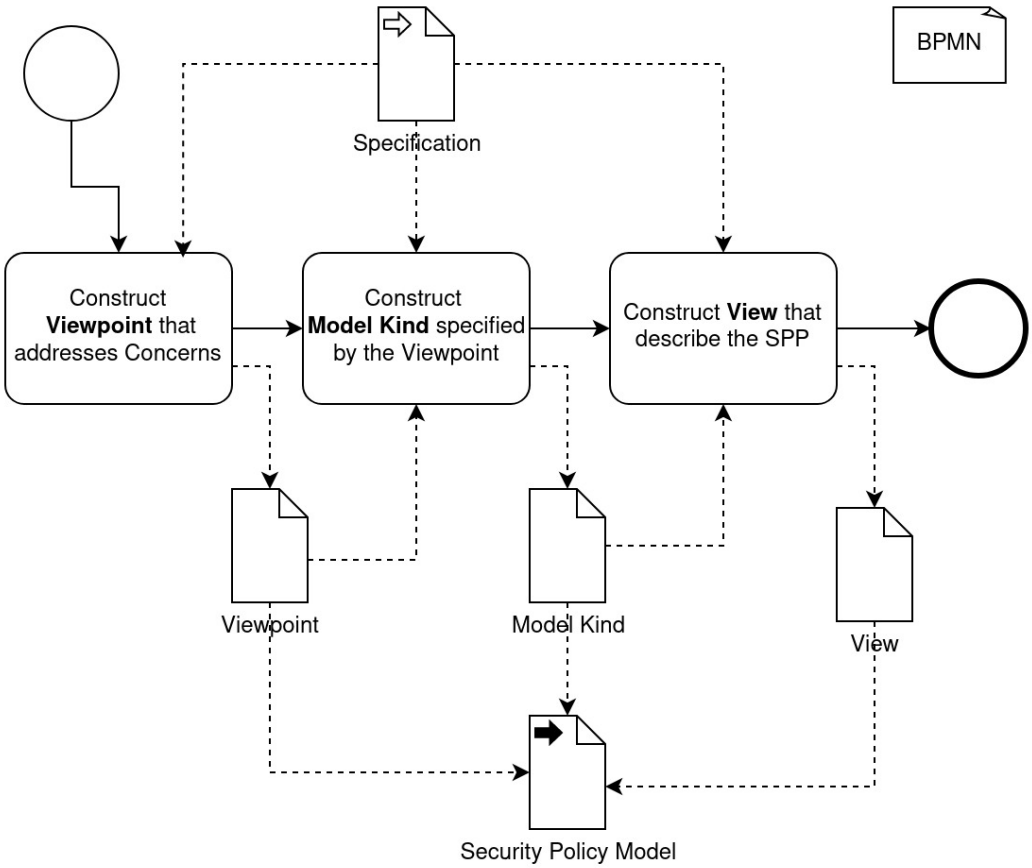

Figure 7.3.: Construction process for the artifact SPM of the Security Policy Meta Model

7.2.3. Automation Model

This section presents the process for constructing the AM artifact. The SPM is the foundation for constructing an AM for the View. A structured approach to automation can take many forms, such as code generation, application specification, or task description. The choice of automation approach depends on the specific *Security Policy* and *Stakeholder Concerns*. Figure 7.4 presents the BPMN process for building the AM artifacts. In the following section, each BPMN task and its corresponding artifacts will be discussed in detail.

Automation Strategy: The first task is the construction of *Automation Strategy* artifacts. This process requires two inputs: *Model Kind* and *Concerns*. The *Concerns* dictate the type of automation appropriate for each *Stakeholder*, while the *Model Kind* serves as the basis for defining the syntactic and semantic elements that require an *Automation Strategy*.

Automation Solution: The second task is to construct the artifact of the *Automation Solution*. With the *Automation Strategy* established, we are now ready to construct the *Automation Solution* in conjunction with the constructed *View* and *Concerns*.

Automation Model: The *Automation Strategy* and *Automation Solution* artifacts form the AM artifact.

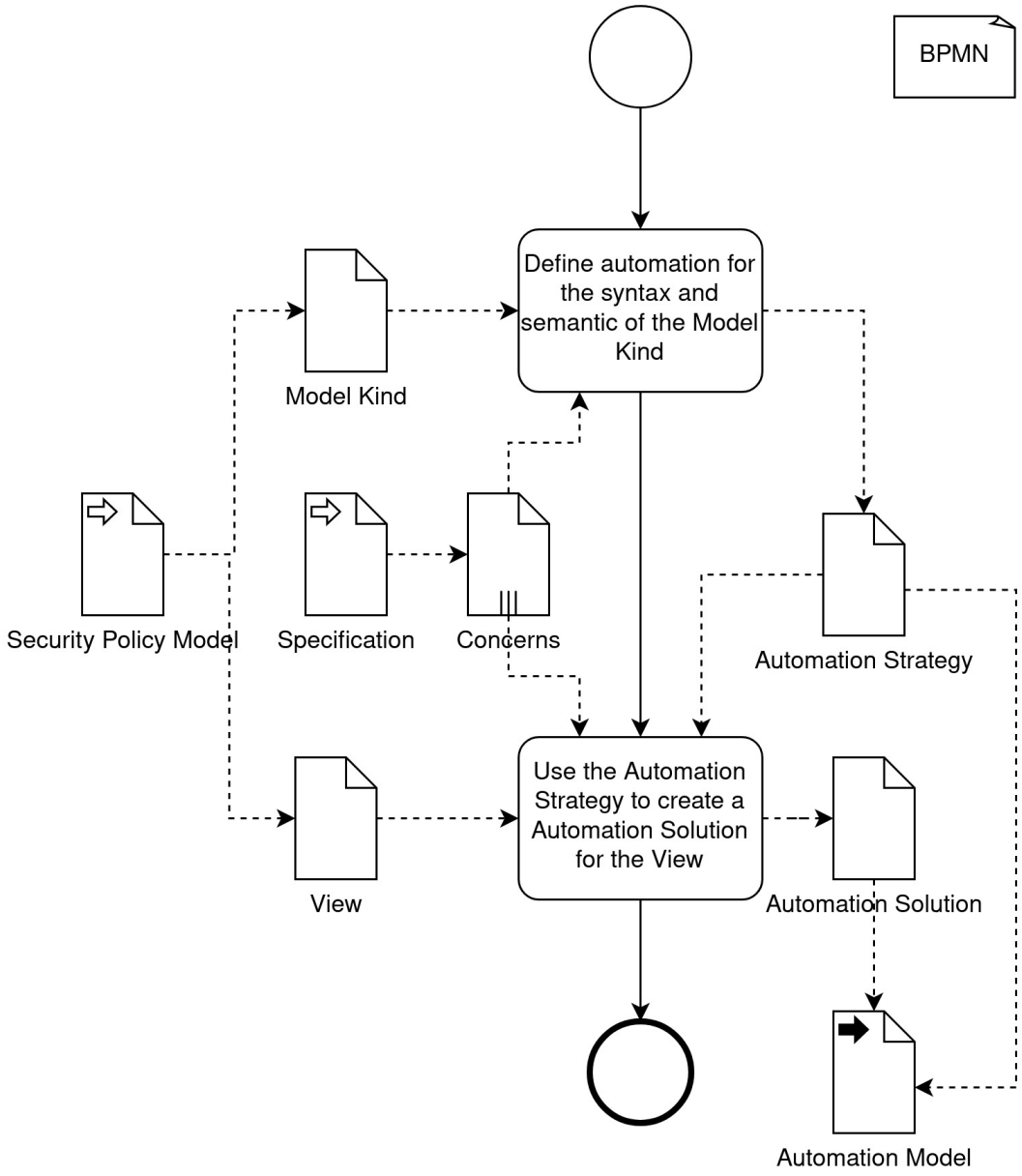

Figure 7.4.: Construction process for the artifact AM of the Security Policy Meta Model

8. Application Examples

Contents

In this chapter, we focus on the application of the proposed construction process of a *Security Policy* Automat Model 7. The process is illustrated by building multiple SPAMs for multiple *Security Policies* and *Stakeholders*.

The first section presents a notation for visually representing a SPAM. This is followed by two sections that introduce the *Security Policies* and *Stakeholders* used in the application examples. Finally, three application examples are presented to visualize the SPMM and the process of constructing SPAMs.

The first example provides a comprehensive visualization of the construction process, for the *Security Policy* **Load Balancer** and the *Stakeholder* **System Administrators**. This example is intended to provide a complete understanding of the construction process.

The second example focuses on sharing customized SPMs among *Stakeholders*. This example presents two SPMs applicable to **Software Developers**, **System Administrators**, and **Software Architects** within the context of the *Security Policy* for a **Zero Trust Network**.

The third example illustrates the versatility of *Automation Strategies* and *Automation Solutions*. It explores the potential for constructing multiple AMs for different *Stakeholders*

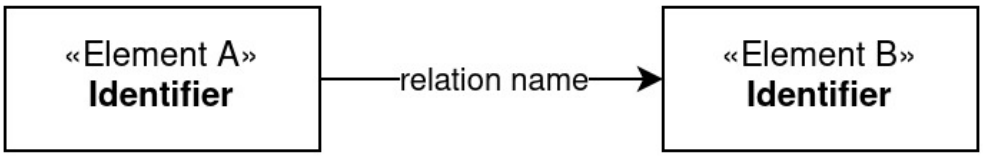

Figure 8.1.: Conceptual example of the Visual Representation

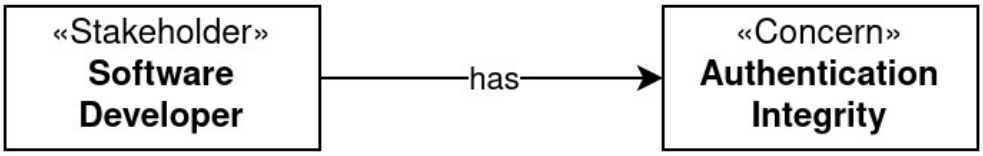

Figure 8.2.: Concert example of the Visual Representation

to automate a single SPM. The automation models are built for the *Software Developer* and *Customer Support Stakeholders*, and the *Security Policy* is *Deny Access by Default*.

8.1. Security Policy Meta Model Visual Representation

We use a notation similar to UML class diagram stereotypes to represent the entities of the model elements of the SPMM. In this notation, the name of the model element is enclosed by '´' and '˘' symbols. The identifier of the model element entity is placed below the name. Relationships between two model elements are represented as unidirectional arrows connecting their respective boxes, with the name of the relationship written on the arrow. Figure 8.1 presents a conceptual example of this notation, while Figure 8.2 illustrates a specific instance with the *Stakeholder Software Developer* and the *Concern Authentication Integrity*.

To simplify the visualization and reduce the number of relationship arrows, we introduce the concept of a 'container box'. This 'container box' groups entities of the same type that share identical relationships. Figure 8.3 presents a conceptual illustration of this notation. Figure 8.4 provides an example that demonstrates this approach with the *Stakeholder Software Developer* and the *Concerns Authentication Integrity* and *System Overload*.

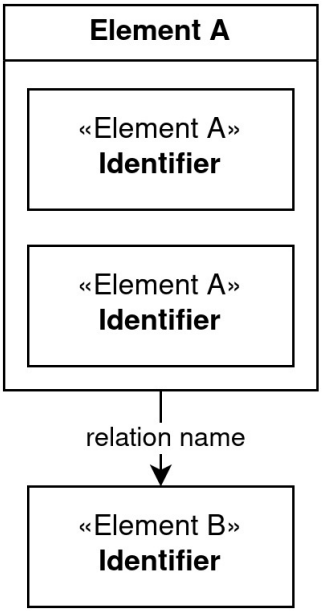

Figure 8.3.: Conceptual example of the Visual Representation

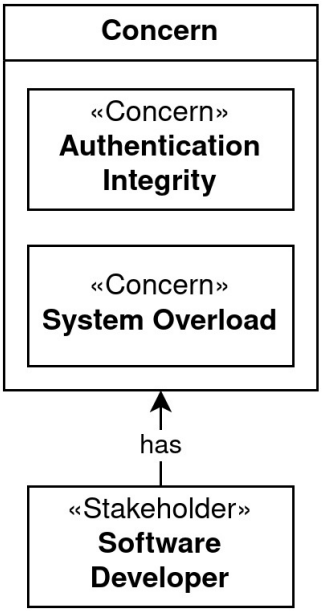

Figure 8.4.: Concert example of the Visual Representation

8.2. Example Security Policies

The *Security Policies* outlined in this section are derived from Sinkovec's Catalog of Security Policies [Sin22]. It is important to note that these *Security Policies* serve only as illustrative examples for our construction process and are not expected nor should be considered comprehensive or complete.

8.2.1. Load Balancer

The *Load Balancer Security Policy* is a preventive measure in microservice architectures that is particularly relevant when an application experiences high demand. It addresses a scenario where a single microservice application, overwhelmed by an increasing volume of requests, faces potential outages, performance degradation, and system failures due to limited resources. This policy involves deploying redundant instances of the microservice managed by a load balancer. The load balancer intelligently distributes incoming requests across available microservice instances based on a defined scheduling strategy. It also dynamically adjusts the number of instances in response to fluctuating demand. Deploying duplicate instances on different hosts, either virtual or physical, further ensures an even distribution of client requests, ensuring application availability and maintaining optimal performance. This approach not only improves system resiliency, but also enhances security by minimizing the risk of service interruptions and potential vulnerabilities that

could be exploited during periods of high load [Sin22].

8.2.2. Zero Trust Network

The *Zero Trust Network Security Policy* assumes that the microservice network is potentially compromised. This approach requires that all actions within the network be rigorously audited and that all data transfers be encrypted. It eschews the concept of implicit trust between services and instead advocates a model in which trust is not assumed, but continuously evaluated and established based on clear, ongoing assessments. This underscores the importance of a proactive, vigilant security posture, particularly in environments where the integrity of the network may be at risk [Sin22].

8.2.3. Deny Access by Default

The *Deny Access by Default Security Policy* is intended to enhance the security of data and services. This requires that all entities are initially denied access to data and services by default. Such a measure is crucial to prevent any unauthorized or unprivileged entities from gaining access, thereby strengthening the security framework against potential breaches or misuse of sensitive information and resources [Sin22].

8.3. Example Stakeholders

The *Stakeholders* outlined in this section are derived from a variety of sources. It is important to note that these *Stakeholders* are only illustrative examples of our construction process and are not intended to be nor should be considered comprehensive or complete.

8.3.1. System Administrator

System Administrators are key players in software systems, primarily responsible for installing, supporting, and maintaining servers and computer systems. They play a critical role in managing and planning for service outages and other technical problems.

System Administrators are an integral part of maintaining a secure computing environment. They implement security measures, monitor compliance, conduct audits, and work with information security professionals. They are also responsible for access controls, security incident response, and disaster recovery planning.

8.3.2. Software Developer

Software Developers are central to the development of software systems, with roles that include coding, design contribution, quality assurance, and documentation. They are essential in translating requirements into functional code. Developers also collaborate across teams and are involved in both initial development and ongoing maintenance, contributing significantly to the success and evolution of the software.

8.3.3. Software Architect

A *Software Architect* plays a key role in the development of software systems, aligning them with business objectives and technical standards. Their role includes defining the software architecture to ensure scalability, resilience and performance, while also serving as a technical leader, mentoring developers and deciding on technology stacks. Effective communication with *Stakeholders* such as customers, managers, and developers is critical to aligning technical design with business needs and user requirements. They are also responsible for creating architectural documentation for development and future maintenance, ensuring compliance with security, privacy, and other non-functional requirements.

8.4. Load Balancer Security Policy Automation Model

This section presents the construction process of the SPAM for the *Security Policy Load Balancer*, focusing on the *Stakeholder System Administrator*. This section is divided into three parts.

Analogous to the construction process of a Security Policy Automation Model, each artifact is presented in a separate subsection. First, the CP of the *Specification* is presented, then the CP of the SPM is presented, and finally the CP of the AM is presented.

8.4.1. Specification

This section presents the CP for the artifact *Specification*. Figure 7.2 illustrates the BPMN process for the artifact *Specification*. In the following section, each BPMN task and its associated artifacts are presented.

Stakeholder

In the first Task the *Stakeholder* artifact is constructed. To accurately reflect *Stakeholders* within the context of *Security Policies*, a more focused description of *Stakeholders* may be necessary. This is particularly important when *Stakeholders* and *Security Policies* originate from different domains. For instance, in our example, both the *Stakeholder* and the *Security Policy* are from the same domain - software engineering - thus a focused definition is not required.

As a result, the *Stakeholder* artifact in this example is the *Stakeholder System Administrator* definition already presented.

Concerns

The second task is to construct the *Concern* artifacts. Below are some example *Concerns* for the *System Administrator Stakeholder* in the context of the *Load Balancer Security Policy*.

Concern: System Overload

The *System Administrators* need to monitor the system load to prevent any potential system failures.

Concern: System Maintenance and Patching

Regular maintenance and patching of the load balancer and service instances is required.

Concern: Deployment of service instances

How is the deployment process of new instances handled?

In the following sections, only the *Concern Deployment of service instances* is used.

Security Policy Perspective

The tired and final task constructs the *Security Policy Perspective* artifact. In this example, we will focus on the previously introduced *Concern*, *Deployment of Microservice Instances* 8.4.1, to construct a *Security Policy Perspective*. The constructed *Security Policy Perspective* is shown below:

Security Policy Perspective: Load Balanced deployment strategy

Implementing a load balancer is an effective solution for handling increased traffic. The redundancy provided by multiple instances of the same service ensures that no single instance is overwhelmed with requests, preventing potential outages or failures. A properly chosen scheduling strategy can ensure the availability of the service. The scheduling strategy determines when service instances are started and stopped.

With the artifacts *Stakeholder*, *Concerns* and *Security Policy Perspective* the artifact *Specification* is constructed and can be used in the next task of the High Level Construction Process of a SPAM.

In Figure 8.5, the constructed artifacts and their relationships are presented, following the visual representation of the SPMM 8.1 introduced earlier.

8.4.2. Security Policy Model

This section presents the process for constructing the SPM artifact. Figure 7.3 illustrates the BPMN process for constructing the SPM artifact. In the following section, each BPMN task and its associated artifacts are discussed in detail.

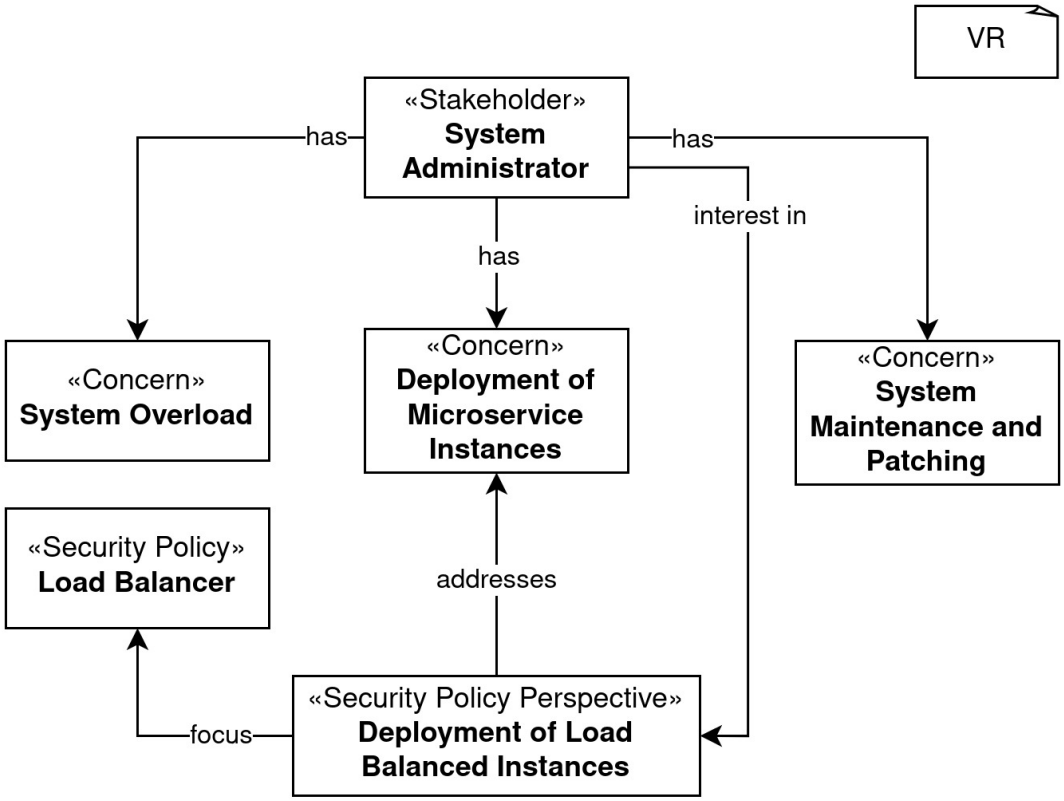

Figure 8.5.: System Administrator Load Balancer - Specification

Viewpoint

The first task constructs the *Viewpoint* artifact. In this example, we revisit the previously introduced *Concern*, *Deployment of Microservice Instances* 8.4.1, to construct a *Viewpoint*. The first *Viewpoint* we construct is the following:

Viewpoint: Container Deployment

This *Viewpoint* involves designing a robust deployment strategy for microservice instances that not only ensures efficient load balancing, but also prioritizes security. This includes creating a secure environment for deploying these instances, implementing proper authentication mechanisms to prevent unauthorized access, and integrating regular auditing processes to identify potential security threats.

Multiple *Viewpoints* can be constructed for a single *Concern*. The following section presents a second *Viewpoint* constructed:

Viewpoint: Load Balancer Efficiency

This *Viewpoint* focuses on the performance of the load balancer. This includes monitoring its ability to effectively distribute incoming requests, add or remove service instances as needed, and distribute these instances across different hosts for optimal distribution.

Model Kind

The second task is to construct the *Model Kind*. In this example, we revisit the previously introduced *Viewpoint*, *Deployment of service instances* 8.4.2, to construct a *Viewpoint*. The BPMN modeling language is used as the basis for our *Model Kind*. In addition, some pre-defined BPMN-Tasks are defined. The inclusion of pre-defined tasks improves the comprehensibility of the *View*.

Pre-defined BPMN-Tasks:

BPMN-Task: Request environment variables

This BPMN-Task details the process of creating the necessary environment variables to initialize new service instances.

BPMN-Task: Start new container

This BPMN-Task details the process of creating a new container using the Docker containerization platform to serve as an instance for the service.

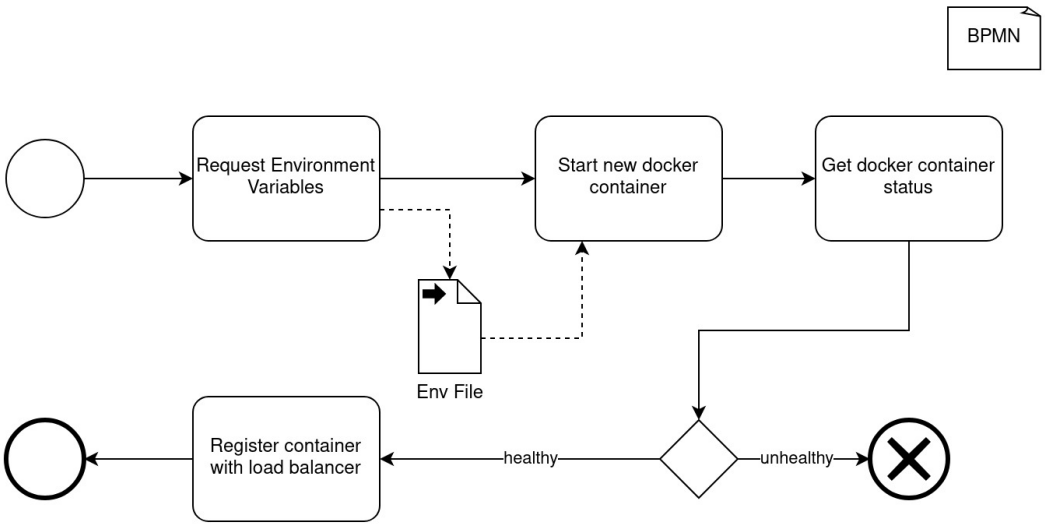

Figure 8.6.: Spawn new service instance on increased load

BPMN-Task: Get container status

This BPMN-Task details the process of querying the status of a container.

BPMN-Task: Register container with the load balancer

This BPMN task details the process of registering a newly created container with the load balancer so that the Load Balancer can direct traffic to the container.

View

The tired and final task is to construct the *View* artifact. With the constructed *Viewpoint* and Model Type, we can now construct a *View* that models the *Security Policy Load Balancer* for the *Stakeholder System Administrator*.

Figure 8.6 illustrates the constructed *View* with the name: *Spawn new service instance on increased load*. This *View* encapsulates the *Security Policy* aspect of spawning new service instances on increased load.

The model outlines the response process to an increased load, beginning with the preparation for creating a new service instance. This involves requesting the necessary environment variables. A new container is subsequently created. Next, the status of the container is queried. Once the container has reached a stable state, the load balancer is notified of the new instance. If the container fails to become healthy, a notification is sent to an external system for further investigation.

In Figure 8.7, the constructed artifacts and their relationships are presented, following

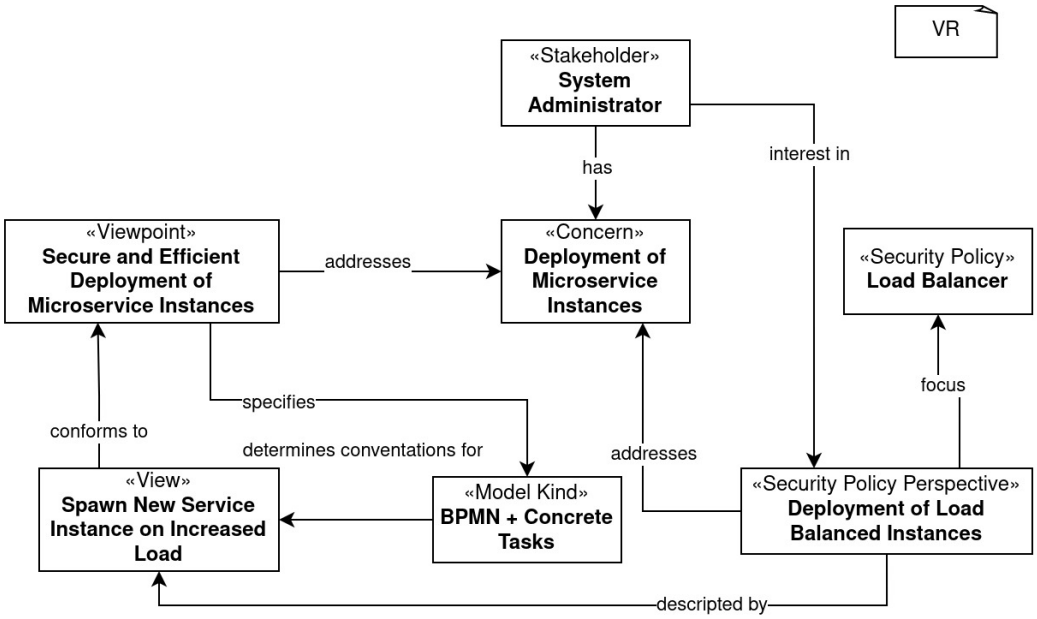

Figure 8.7.: System Administrator Load Balancer - Specification and SPM

the visual representation of the SPMM 8.1 introduced earlier. This visual representation also includes the relevant artifacts from the construction process of the *Specification* artifact 8.5.

8.4.3. Automation Model

In this section the CP for constructing the artifact AM is presented. The SPM is foundational in construction an AM for the *View Spawn new service instance on increased load* 8.6.

In our example, the *View* is interpreted through the lens of the *System Administrator*'s *Concerns*. This perspective encapsulates the control flow for spawning new service instances. Our proposed automation involves triggering a script in response to load increases detected by the load balancer. This automation is constructed by defining a series of bash scripts, each tailored to address the particular syntactic and semantic aspect of the *Model Kind* 8.4.2 associated with the *View* 8.6.

Figure 7.4 presents the BPMN process for the construction of the AM artifacts. The following sections describe each BPMN task and the corresponding artifacts.

Automation Strategy

The first task constructs the *Automation Strategy*. For the *Spawn new service instance on increased load* 8.6 *View*, the *Automation Strategy* example is limited to elements used within that *View*. For each BPMN task explicitly defined in the *Model Kind*, a corresponding bash script is constructed.

The Listing 8.1 provides an *Automation Strategy* for the *BPMN-Task: Request environment variables* 8.4.2. This script fragment executes a bash script that generates a file named `.env` located in the current directory. This file typically contains key-value pairs, providing a convenient way to pass configuration settings into the container.

```
bash ./deployment/scripts/generate-env-file.sh
```

Source Code 8.1: Automation Strategy for BPMN-Task: Request environment variable

The Listing 8.2 provides an *Automation Strategy* for the *BPMN-Task: Start new container*. This script fragment creates and starts a container based on a specified docker image and assigns the container ID to the variable `CONTAINER_ID`. The `docker run` command is used to run the container. The `-env-file .env` option specifies that the environment variables for the container are read from a file named `.env` in the current directory. The `-detach` flag starts the container in the background. The `DOCKER_IMAGE` is a variable containing the name (and optionally the tag) of the Docker image.

```
CONTAINER_ID=$(docker run --env-file .env --detach $DOCKER_IMAGE)
```

Source Code 8.2: Automation Strategy for BPMN-Task: Start new container

The Listing 8.3 provides an *Automation Strategy* for the *BPMN-Task: Get container status* 8.4.2. This script fragment writes the status of the container into the variable `STATUS`. By piping the output of the `docker ps` command with the variable `CONTAINER_ID` as a parameter to the `get_status` command, the status is extracted.

```
STATUS=$(docker ps | grep "$CONTAINER_ID" | get_status)
```

Source Code 8.3: Automation Strategy for BPMN-Task: Get container status

The Listing 8.4 provides an *Automation Strategy* for the *BPMN-Task: Register container with the load balancer* 8.4.2. This script fragment sends an HTTP POST request to a specified URL, in this case the domain of the load balancer container. The request body for this POST request is taken from the value of the `CONTAINER_ID` environment variable.

```
http post http://load-balancer.local/register << \$CONTAINER_ID
```

Source Code 8.4: Automation Strategy for BPMN-Task: Register container with the load balancer

Additionally, bash scripts have been constructed for the BPMN events used in the *View*, specifically for *None-Start*, *Cancel-End*, and *End*. The Listing 8.5 provides an *Automation Strategy* for the *BPMN-Event: None-Start*. This script fragment indicates that the script should use the bash interpreter.

```
#!/bin/bash
```

Source Code 8.5: BPMN-Event None-Start

The Listing 8.6 provides an *Automation Strategy* for the *BPMN-Event: Cancel-End*. This script fragment exits the script and returns a status code indicating an error or abnormal termination of execution.

```
exit 1
```

Source Code 8.6: BPMN-Event Cancel-End

The Listing 8.7 provides an *Automation Strategy* for the *BPMN-Event: End*. This script fragment exits the script and returns a status code indicating a successful execution.

```
exit 0
```

Source Code 8.7: BPMN-Event End

Automation Solution

The second task is to construct the *Automation Solution*. In this example, this involves composing script fragments that match the control flow defined in the View. Listing 8.8 demonstrates this composed script. It's important to note that the structure of the construction may vary significantly depending on the specific *Concerns*, *Automation Strategy*, and *View*.

```
#!/bin/bash

bash ./deployment/scripts/generate-env-file.sh

CONTAINER_ID=$(docker run --env-file .env --detach $DOCKER_IMAGE)

STATUS=$(docker ps | grep "$CONTAINER_ID" | get_status)

if $STATUS !== "healthy"; then
  exit 1;
fi

http post http://load-balancer.local/register << $CONTAINER_ID

```

```
exit 0
```

Source Code 8.8: Automation Solution: Create new container instance and register with the Load Balancer

The *Automation Strategy* and *Automation Solution* artifacts construct the AM artifact.

In Figure 8.8, the constructed artifacts and their relationships are presented, following the visual representation of the SPMM introduced earlier. This visual representation also includes the relevant artifacts from the *Specification* and *Security Policy* artifacts.

8.5. Zero Trust Network Security Policy Model

In this section, we focus on the ability of our SPMM to support the creation of SPMs tailored to the *Concerns* of multiple *Stakeholders*. Following the CP outlined in Chapter 7, this example focuses on the development of the *Specification* and SPM artifacts. The *Stakeholders* involved are the *Software Developer*, the *System Administrator*, and the *Software Architect*, and the *Security Policy* in question is the *Zero Trust Network*. The first SPM is tailored to the *Software Developer* and *Software Architect*, while the second SPM is tailored to the *System Administrator* and *Software Architect*.

The first subsection describes the construction of the *Specification* artifacts, followed by the second subsection, which describes the construction of the SPM artifacts.

8.5.1. Specification

In this section, the two *Specification* artifacts are constructed. First, the *Stakeholder* artifacts are constructed. Second, the *Concern* artifacts are constructed for the *Stakeholders* with the assignment for which *Stakeholders* have the *Concerns*. Next, the *Concerns* are grouped into two groups. Then, for each group, the *Security Policy Perspective* artifact is constructed.

Stakeholder

Similar to the first example for the *Stakeholder Software Developer*, *System Administrator*, and *Software Architect*, it is not necessary to construct more specific *Stakeholders*, and the current *Stakeholders* can be used directly as *Stakeholder* artifacts.

Concerns

For the *Stakeholder Software Developer*, *System Administrator*, and *Software Architect* the following *Concerns* are defined:

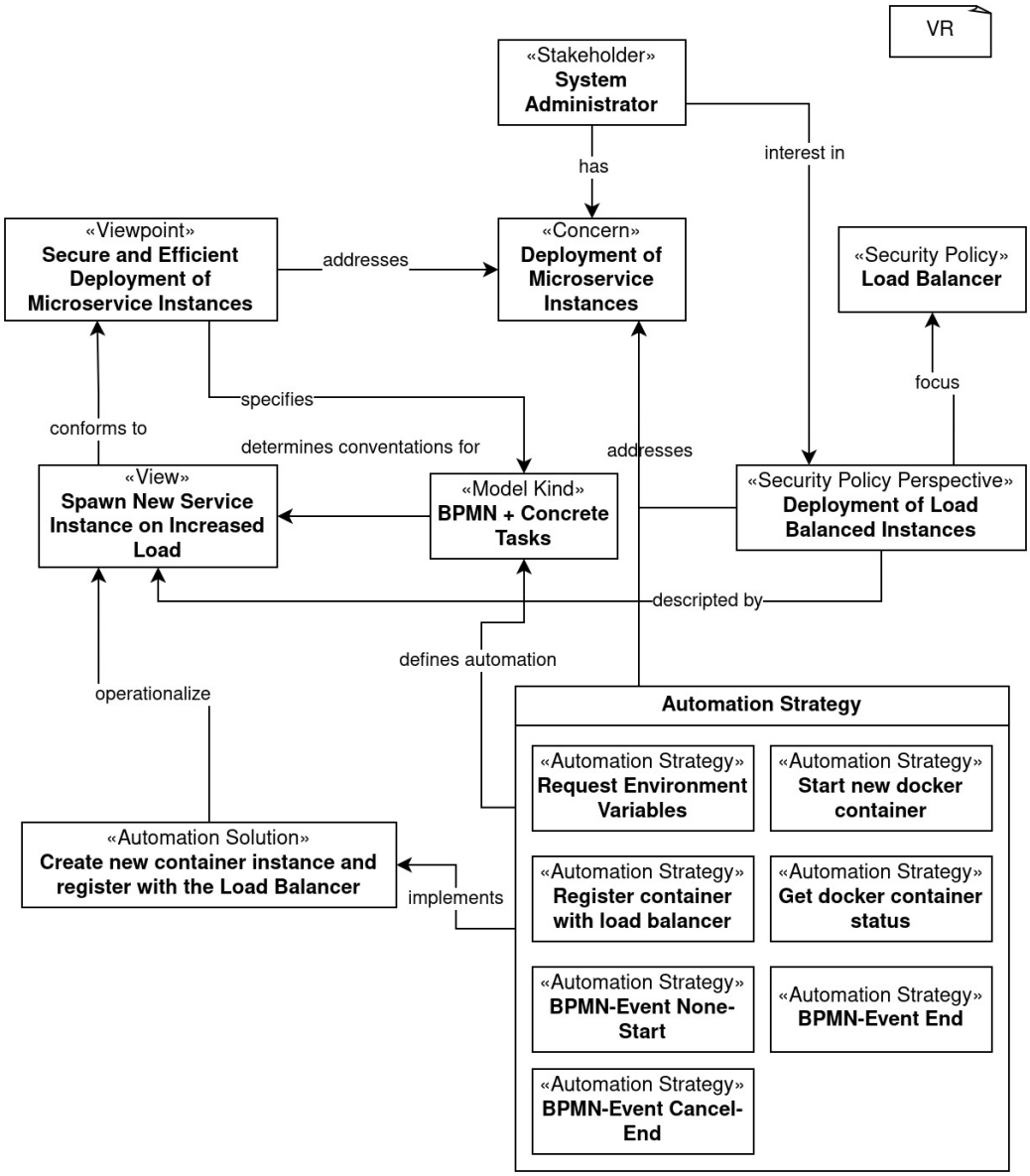

Figure 8.8.: System Administrator Load Balancer - Specification and SPM and AM

Concern: Authentication Integrity

This *Concern* involves ensuring the validity and security of user authentication mechanisms. It includes developing secure login processes, managing user credentials, and implementing multi-factor authentication. Ensuring that these mechanisms are robust and resistant to breaches is critical.
(Software Developer)

Concern: Developer Training

The ongoing education and skill development of the software development team. This includes providing training on new technologies, coding best practices, and updates to software development methodologies.
(Software Architect)

Concern: Service Segmentation

Separation of services into distinct, manageable, and secure components. This involves designing the system architecture to logically separate services.
(Software Developer, System Administrator, Software Architect)

Concern: Policy Enforcement

Implement and maintain organizational and technical policies. This includes establishing security protocols, data management policies, and operational procedures.
(System Administrator, Software Architect)

Concern: Monitoring and Logging

The continuous monitoring of system operations and recording of activities. The focus is on detecting anomalies, performance issues, and potential security breaches.
(System Administrator, Software Architect)

Concern: Incident Response Readiness

Prepare to manage potential system incidents. This includes developing incident response plans, establishing communication protocols, and regularly testing response procedures.
(System Administrator, Software Architect)

The *Stakeholder Software Architect* shares all *Concerns* with all other *Stakeholders* except *Authentication Integrity* and has the *Concern Developer Training*. The *Stakeholder Software Developer* has the *Concerns Authentication Integrity* and *Service Segmentation*. The *Stakeholder System Administrator* has the *Concerns Policy Enforcement*, *Monitoring*

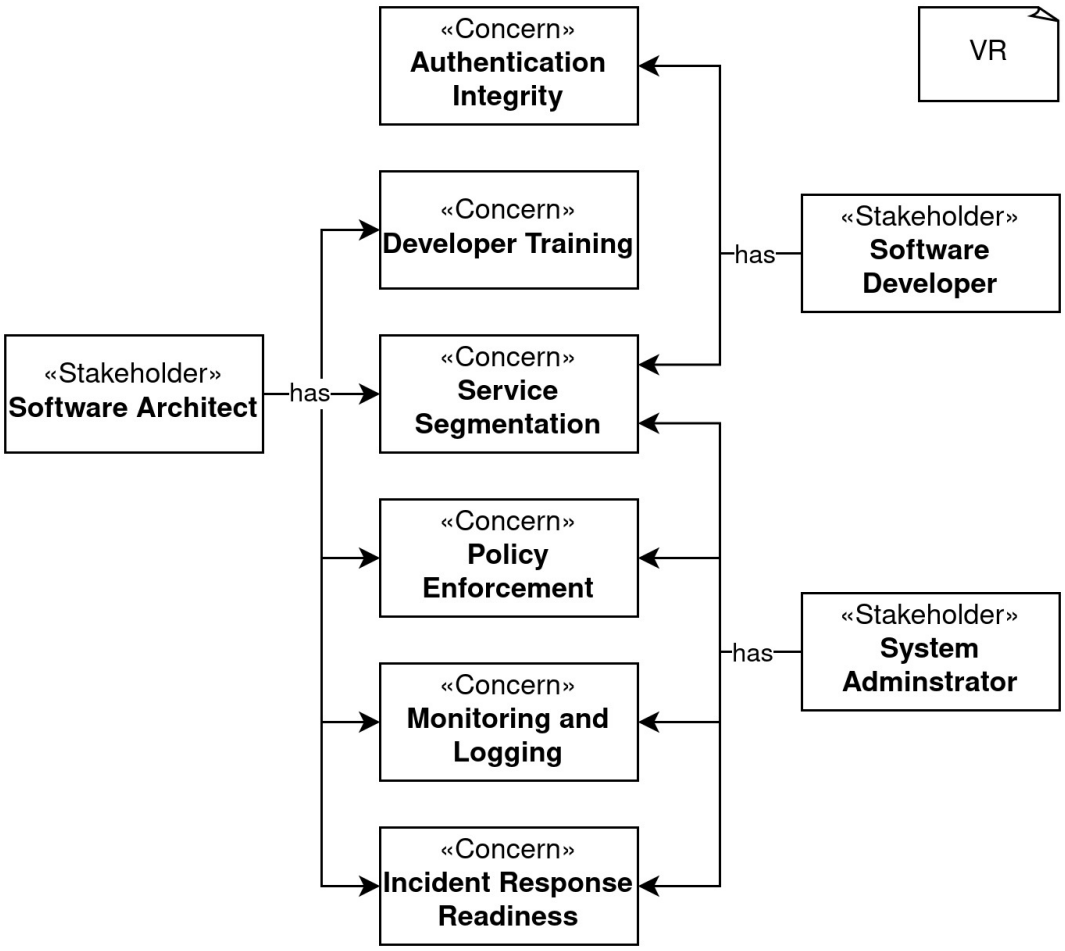

Figure 8.9.: Concern and there relationships to Stakeholder

and Logging, and *Incident Response Readiness*. An overview of all identified *Concerns* and their relationships to *Stakeholders* is presented in Figure 8.9. We can define two groups of these *Concerns*. The first group includes the *Concern Service Segmentation*, which is associated with the *Stakeholders*: *Software Architect*, *Software Developer* and *System Administrator*. The second group includes the *Concerns Policy Enforcement*, *Monitoring and Logging* and *Incident Response Readiness*, associated with the *Stakeholders*: *Software Architect* and *System Administrator*.

Security Policy Perspective

With the two groups of *Concerns* defined, we can now construct two *Security Policy Perspectives* that address the *Stakeholder*'s *Concerns*.

For the first group, including the *Concern Service Segmentation*, the following *Security*

Policy Perspective can be constructed:

Security Policy Perspective: Zero Trust Network with Service-Oriented Segmentation

This perspective focuses on the isolation and secure integration of services within the system. It emphasizes the need for clear boundaries and communication protocols between services to enhance security. The perspective advocates for compartmentalization and the principle of least privilege, ensuring that each service operates with the minimum necessary access rights, thereby limiting the potential impact of security breaches.

The second group, which includes *Concerns* such as *Policy Enforcement*, *Monitoring and Logging*, and *Incident Response Readiness*, the following *Security Policy Perspective* can be constructed:

Security Policy Perspective: Zero Trust Network and proactive Compliance and Response Management

This perspective is designed to address the operational aspects of *Zero Trust Network*, focusing on proactive policy enforcement, vigilant monitoring, and efficient incident response. The perspective encompasses a comprehensive approach to security, integrating continuous monitoring with dynamic *Security Policy* enforcement and rapid incident response capabilities. This ensures a robust defense against security threats and efficient management of potential breaches.

In Figure 8.10, the constructed artifacts and their relationships are presented, following the visual representation of the SPMM introduced earlier. This visualization also includes an indicator for the two *Specification* artifacts, A and B. The artifact *Stakeholder Software* should be considered in both *Specification* artifacts.

8.5.2. Security Policy Model

In this subsection, the SPM for *Specifications* A and B is constructed. First, the SPM for *Specification* A is constructed. Then we construct the SPM for *Specification* B.

Specification A

In this section, the tailored SPM is constructed for the *Software Developer* and *Software Architect Stakeholders*. First, the *Viewpoint* is constructed, followed by the *Model Type*, and finally the *View*.

Viewpoint: The *Viewpoint* for *Specification* A, which addresses the *Zero Trust Network with Service-Oriented Segmentation Security Policy Perspective*, is constructed as follows:

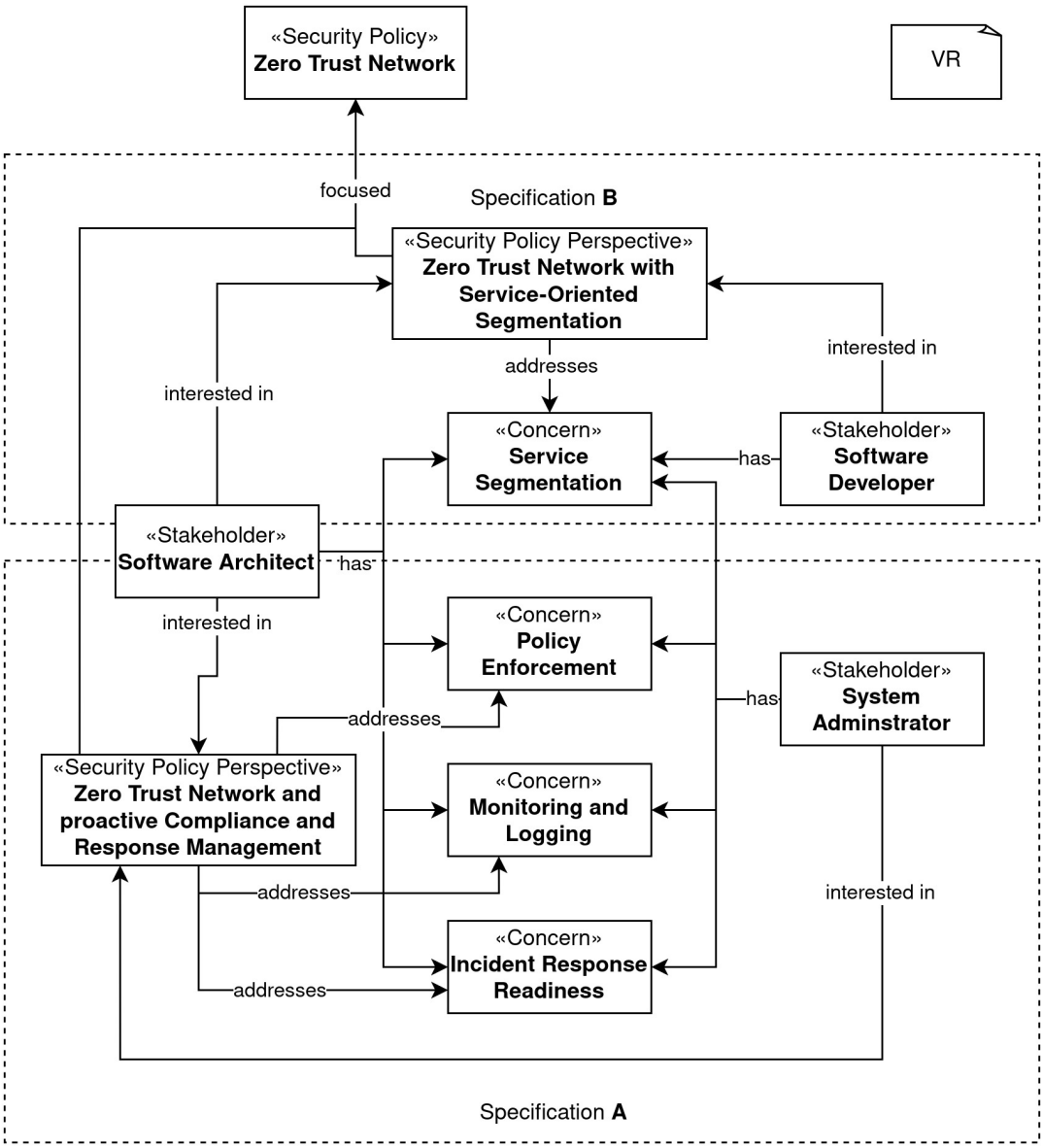

Figure 8.10.: Zero Trust Network Security Policy Model - Specifications

Viewpoint: Enhanced Security through service Isolation

Emphasizes the importance of segregating services to prevent unauthorized access and data breaches. This supports the Zero Trust philosophy by treating each service as a potential threat vector, requiring strict access controls and auditing. By isolating services, the network reduces the risk of widespread system compromise if a single service is compromised. This perspective aligns with *Stakeholder Concerns* and focuses on service-level security, ensuring that each service is independently secure and able to defend against potential threats.

Model Kind: A novel model language is introduced for the *Model Kind*. First, we present the syntax of this model language, followed by its semantics. In accordance with our *Viewpoin*, the model language consists of three syntactic elements: Container, Pod, and Network Communication, as shown in Figure 8.11.

Container

This element symbolizes a Docker container identified by a unique name.

Pod

This element represents a Kubernetes pod, which can contain multiple containers.

Network Communication Arrow

This signifies the communication link between two Containers.

Figure 8.12 illustrates the network communication between two Containers.

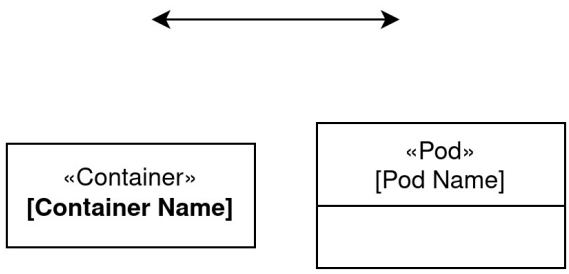

Figure 8.11.: Pod Deployment Diagram Syntax

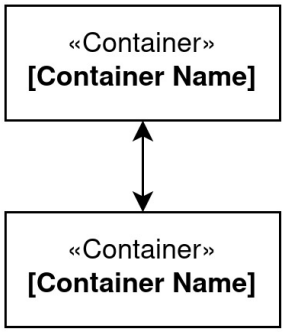

Figure 8.12.: Represents the network communication between contains

A key feature of the *Pod* element is its ability to contain multiple *Containers*, mirroring the Kubernetes concept where a pod can contain multiple Docker containers. This is represented visually in Figure 8.13.

It's also important to note that the communication arrow, which indicates interaction between two *Containers*, is only applicable under two conditions: either the *Containers* are inside the same *Pod*, or a *Container* is partially drawn outside the *Pod*. This particular illustration is shown in Figure 8.14.

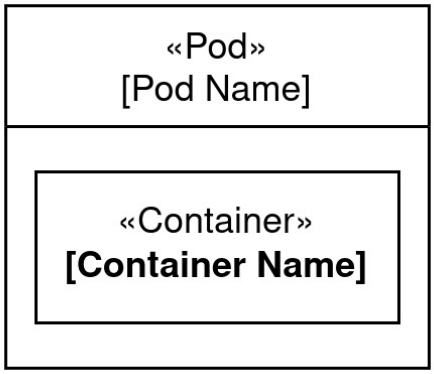

Figure 8.13.: Represents a named Pod that contains one or multiple Container that can NOT communicate with other Pods

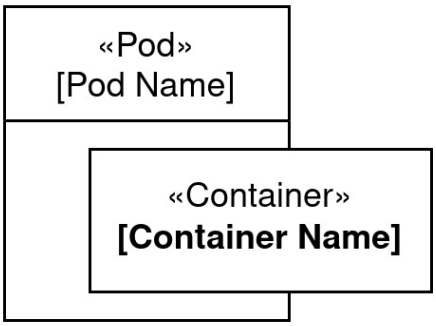

Figure 8.14.: Represents a named Pod that contains one or multiple Container that CAN communicate with other Pods

View: Using the *Viewpoint* artifact and *Model Kind*, we can now construct a *View*. In Figure 8.15 we show the constructed *View*. Where two *Pods* are defined with an application *Container* that communicates to a TLS proxy *Container* that encrypts incoming and outgoing network communication. The TLS proxy *Containers* in turn communicate with each other using encryption.

Specification B

In this section, the SPM is constructed for the *System Administrator* and *Software Architect Stakeholders*. First, the *Viewpoint* is constructed, followed by the *Model Kind* and *View*.

Viewpoint: The *Viewpoint* for *Specification* B, which addresses the *Zero Trust Network and Proactive Compliance and Response Management Security Policy Perspective*, is constructed as follows:

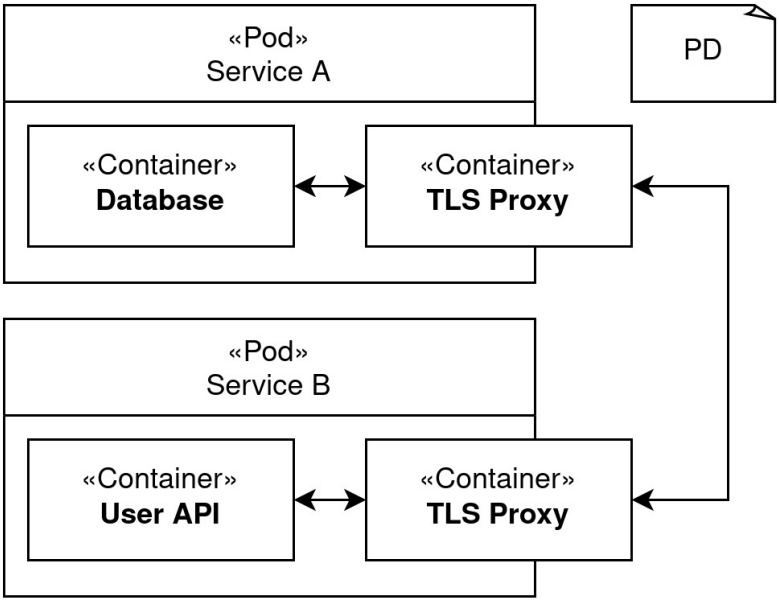

Figure 8.15.: Generic Pod Deployment - modeling the *Security Policy* Zero Trust Network

Viewpoint: Comprehensive Network Security and Incident Management

It incorporates the principles of proactive monitoring, rapid response to security incidents, and strict policy enforcement. It integrates elements of real-time network monitoring, alert mechanisms for potential security breaches, and rapid incident response strategies. It is designed to ensure that any deviation from established security protocols is quickly identified and addressed.

Model Kind: The *Model Kind* uses the same approach as in the first example. The *Model Kind* is based on the BPMN modeling language with a set of pre-defined BPMN tasks that can be used in the construction of the *View*. The following pre-defined BPMN tasks are defined:

BPMN-Task: Send Notification

This BPMN task details the process of sending a notification to inform an external system about an detected unencrypted traffic

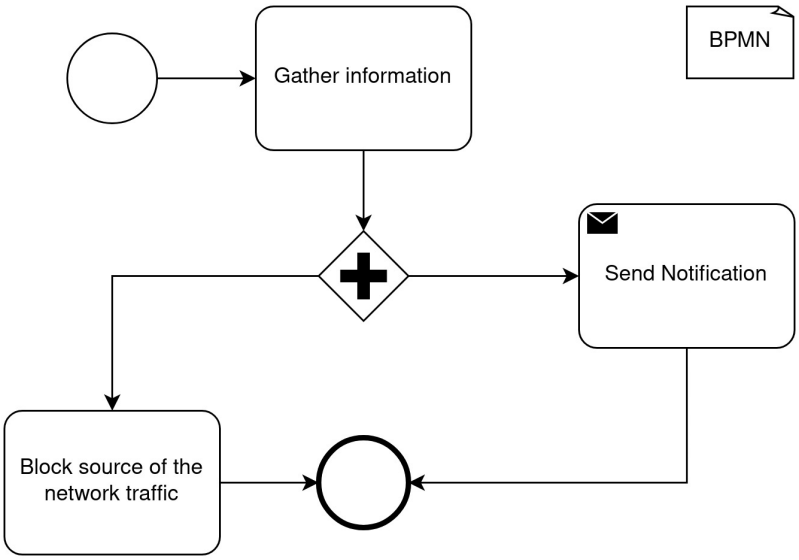

Figure 8.16.: Zero Trust Network *Security Policy View* for Specification B

BPMN-Task: Block source of the network traffic

This BPMN task details the process of blocking the source of the unencrypted traffic in the network,

BPMN-Task: Gather information

This BPMN task details the process of collection information about the origin and destination of the unencrypted traffic

View: With the constructed *Viewpoint* and *Model Kind*, we can now construct the *View*. Figure 8.16 shows the constructed *View*. The *View* describes the process that should be performed when unencrypted traffic is detected in the network. The first step is to determine the origin and destination of the unencrypted traffic. This is followed by the parallel execution of blocking the source of the traffic in the network and sending a notification to inform about the *Security Policy* violation.

8.6. Deny Access by Default Automation Model

In this section, we will focus on the ability of the SPMM to be used to construct multiple AMs for the same SPM. In this example, the *Security Policy Deny Access by Default* and the *Stakeholders System Administrator*, and *Customer Support* are used. First, the *View*, *Model Type*, and *Concerns* are briefly introduced. This is followed by the construction of

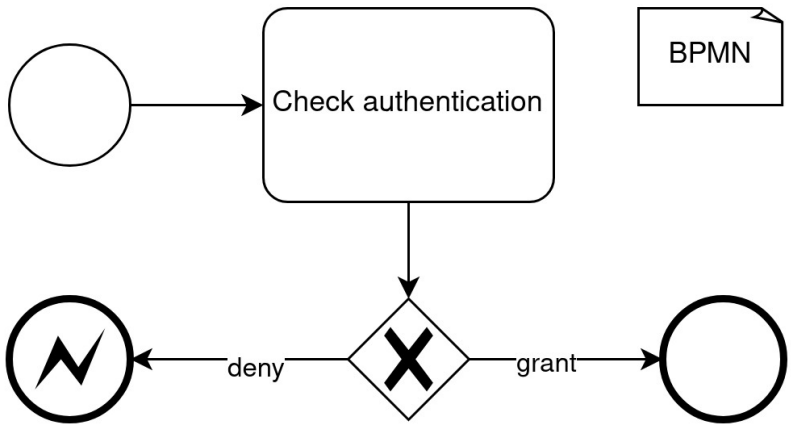

Figure 8.17.: View for the Security Policy Deny Access by Default

the AM for each *Stakeholder*.

8.6.1. Partial Specification and Security Policy Model

This subsection briefly introduces the artifacts needed to construct the AM. It starts with the *Concern*, followed by the *Model Type* and *View*.

Concern: The common *Concern* of all *Stakeholders* is the following:

Concern: Handle unauthorised and authorised access
What to do in the event of unauthorized or authorized access

Model Kind: Similar to the *Model Kinds* from the previous examples, the *Model Kind* is based on the model language BPMN with one predefined BPMN-Task:

BPMN-Task: Check authentication
This BPMN-Task details the process of verifying the authentication.

View: With the *Concern* and *Model Kind*, a simple *View* can be constructed. The *View* models the aspect of how to proceed in the case of unauthenticated and authenticated access to a system. In case of an unauthenticated access, the process ends with an error state. In case of an authenticated access, the process ends with a success state. Figure 8.17 shows the *View*.

8.6.2. Automation Model

This subsection presents the three AMs for each *Stakeholder*. String with the AM for the *System Administrator*, followed by the *Customer Support*.

System Administrator Automation Model

For the *System Administrator* the automation of the *Security Policy* aspect is to modify the application deployment. Firstly the *Automation Strategy* is constructed followed by the *Automation Solution*.

Automation Strategy: Similar to the example in Chapter 6, a statement is defined for each BPMN event and task.

BPMN-Event: Start

Incoming network requests are redirect to the authentication middleware.

BPMN-Event: End

The origin network request is redirected to the origin destination.

BPMN-Event: Error End

An error response is sent to the source of the network request.

BPMN-Task: Check authentication

A network middleware that determine if the request includes basic HTTP authentication.

Automation Solution: To effectively implement this automation, *Stakeholders* must configure application deployment to align with the *Automation Strategy*. This example assumes the use of Docker, a containerization platform, for application deployment. To manage traffic routing to the container, a reverse proxy container such as Traefik is used. Traefik facilitates the configuration of network middleware and traffic redirection through label-based settings for each container.

Traefik is an open source reverse proxy and load balancer designed to simplify the routing of traffic to different services. It integrates seamlessly with containerized environments such as Docker, enabling dynamic configuration and automatic discovery of services.

The Listing 8.9 shows an example of a Docker Compose configuration to implement the Automation Strategy.

This Docker Compose configuration specifies a single service. The service, named "service", is based on the "node:lts" image. It contains labels for integration with Traefik, a reverse proxy.

The first label defines a rule for the Traefik HTTP router, specifying that requests to the hostname service.local should be routed to this service. The second label specifies the use of a middleware called "auth" for this routing. The third label defines the "auth" middleware to use basic authentication, with credentials set to "user:password".

In summary, this configuration sets up a Node.js service that is accessible at service.local and is protected by basic authentication.

```
services:
  service:
    image: node:lts
    labels:
    - "traefik.http.routers.node.rule=Host(`service.local`)"
    - "traefik.http.routers.node.middlewares=auth"
    - "traefik.http.middlewares.auth.basicauth.users=user:password"
```

Source Code 8.9: docker-compose.yaml

Customer Support Automation Model

For the *System Administrator* the automation of the *Security Policy* aspect is to use an application. Firstly the *Automation Strategy* is constructed followed by the *Automation Solution*.

Automation Strategy: Similar to the example in Chapter 6, an actionable statement is defined for each BPMN event and task.

BPMN-Event: Start

Open the Customer relationship management (CRM) application.

BPMN-Event: End

Continue processing the customer request.

BPMN-Event: Error End

Reject the customer request.

BPMN-Task: Check authentication

Enter the customer details into the CRM application and check if the customer exists.

Automation Solution: Following the approach from the example in Chapter 6. A set of instructions for the *Stakeholder* can be defined:

1. Open the Customer relationship management (CRM) application.
2. Enter the customer details into the CRM application and check if the customer exists.
   a) Continue processing the customer request.
   b) Reject the customer request.

9. Discussion

Contents

This chapter provides a detailed discussion of the findings from our CP of SPAMs. First, we discuss the insights gained during the development of the SPMM and the CP of SPAMs. This is followed by a proposal for an evaluation process for feature research due to the limited nature of the application examples conducted. This process is necessary because constructing multiple SPAMs is not enough to assess the robustness of our SPMM.

9.1. Application Example Discussion

This section delves deeper into our SPMM. In the first part we will discuss the advantages of constructing a SPAM tailored towards multiple *Stakeholders*. Subsequently, we explore the feasibility and advantages of constructing multiple AMs that implement same SPM. We then examine the limitations and shortcomings of our SPMM. In the second part, we address our research questions and explain how our SPMM contributes to answering them.

9.1.1. Stakeholder Grouping

In the Chapter 8 our second example 8.5 demonstrated the feasibility of creating a SPM tailored to multiple *Stakeholders*. This is achieved by categorizing *Stakeholders*' *Concerns* in a way that allows these *Concerns* to be shared among them. When *Stakeholders* possess similar domain knowledge, one SPM can be developed for multiple *Stakeholders*. This capability of our SPMM significantly reduces the number of SPMs needed to meet *Stakeholders*' requirements.

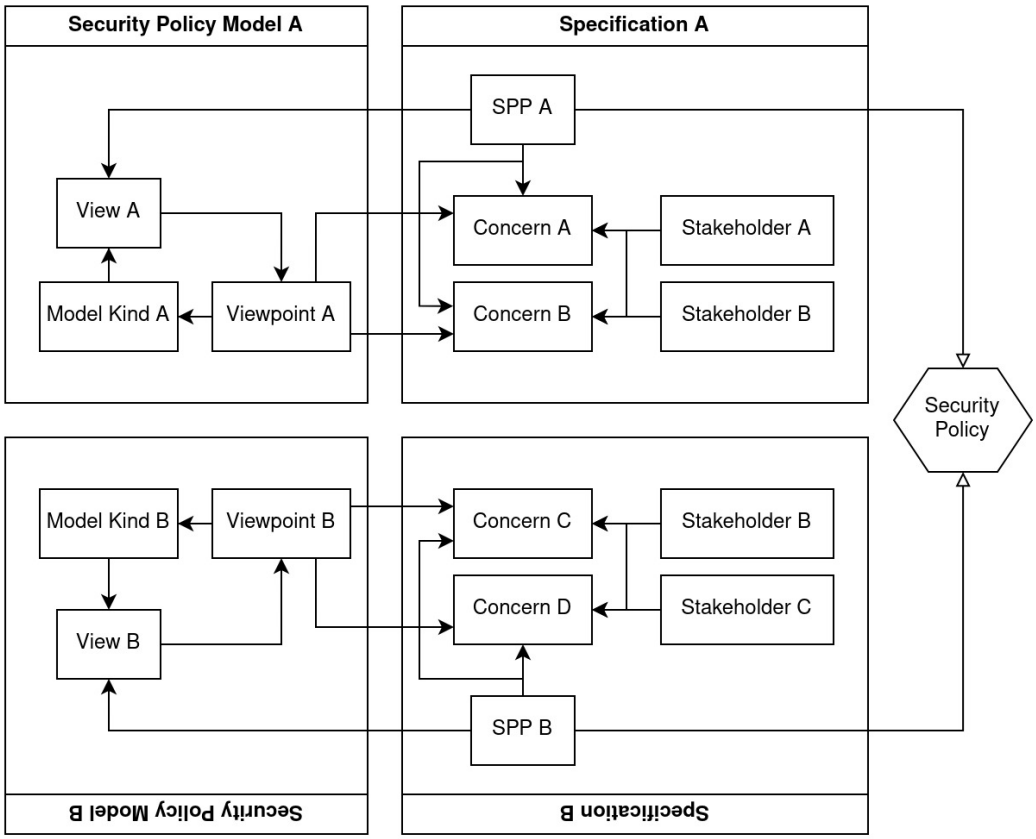

Figure 9.1.: Stakeholder grouping conceptional example

Conceptual Example: Figure 9.1 provides a conceptual example. In this conceptual example, we show how to potentially reduce the number of SPMs that need to be constructed for a given security policy. *Stakeholder* A has *Concerns* A and B. *Stakeholder* C has *Concerns* C and D. Notably, *Stakeholder* B shares *Concerns* B and C with *Stakeholders* A and C.

In this instance, constructing *Security Policy Perspectives* A and B, which respectively address *Concerns* A and B, as well as *Concerns* C and D, would suffice. This results in the definition of only two *Specifications*, A and B. With only two specifications, it would be necessary to construct only two SPMs, A and B. Although this is a highly syntactic example, it demonstrates the potential to reduce the number of SPMs required for a given *Security Policy*.

Concert Example: In a real-world scenario, this approach could be highly advantageous. If an organization or software project has S many *Stakeholders*, and on average, every

Stakeholder has C many *Concerns*, this could lead to the creation of $SPP = S * C$ many *Security Policy Perspectives* or more. By this simplification, the amount of *Specifications*, and therefore SPMs, can be reduced.

For instance, consider an organization with 100 *Security Policies* (SP), 20 *Stakeholders* (S) and on average 3 *Concerns* (C). Traditionally, this would necessitate creating $M = SP * S * C = SP * SPP = 6000$ SPMs, an impractical endeavor. Furthermore, the need to construct and implement 6000 corresponding AMs exacerbates this impracticality. However, by grouping *Stakeholders*, it becomes feasible to drastically reduce the number of required SPMs and, consequently, AMs.

9.1.2. Tailored Automation Models

The third example in Chapter 8, demonstrates the feasibility of construction multiple AMs for a single SPM. This approach is significant in its capacity to address diverse *Concerns* pertinent to various *Stakeholders*. Each AM, while grounded in the same underlying SPM, is tailored to meet the specific *Concerns* of different *Stakeholders*. This customization facilitates a more targeted and effective implementation of *Security Policies*, ensuring that the unique needs and *Concerns* of each *Stakeholder* are adequately addressed. The development of these tailored AMs underscores the adaptability and flexibility of the SPMM, showcasing its utility in varied contexts and its ability to be reconfigured to suit distinct operational scenarios.

9.1.3. Limitations

Analogous to the Concept Model developed by Rostami et al.[RKG22], our SPMM has been tested with only a limited range of *Security Policies* and *Stakeholders*. Therefore, it cannot be conclusively determined whether our SPMM is universally feasible or primarily suited to the specific *Security Policies* and *Stakeholders* examined. This uncertainty underscores the critical importance of implementing our proposed evaluation process. Such an evaluation is essential to ascertain whether our SPMM is broadly applicable or if its utility is confined to certain types of *Security Policies*.

9.1.4. Conclusion

In conclusion, it can be asserted that the construction of a SPM, coupled with an automation approach tailored to *Stakeholder Concerns*, is feasible using our proposed SPMM for the selected *Security Policy*.

Regarding the constructed *Automation Strategy* and *Automation Solution*, it is evident that the application of basic code generators for the *Automation Strategy* is a viable option. However, as previously mentioned in earlier chapters this approach is not the exclusive method for developing *Automation Strategies* and *Automation Solutions*. Espachily when an *Automation Model* for non tecnicly *Stakeholders* is constructed. This *Stakeholder* have *Concerns* whereby the automation of *Security Policies* is more like the utilising of tools to automation there manuel tasks.

9.2. Research questions findings

This section discusses the findings that help answer our research questions:

(RQ1) - *What are the key aspects of security policies in software systems, and how are they structured?*

The SPMM, introduced in the master's thesis, plays a pivotal role in the representation and management of security policies in software systems. The SPMM encapsulates the entirety or select aspects of a security policy, providing a comprehensive framework for understanding and implementing these security policies in a structured manner. Its significance in representing key aspects of security policies in software systems is multifaceted and can be understood through various dimensions of software security and policy management.

Firstly, the SPMM serves as a foundational blueprint that guides the development and enforcement of security policies. It provides a structured approach to defining the rules and principles that safeguard system integrity, confidentiality, and availability. By modeling security policies at a meta-level, it offers a higher abstraction, allowing for the encapsulation of complex security requirements in a manageable and understandable format. This higher-level abstraction is crucial in translating intricate security needs into actionable security policies.

Secondly, the SPMM enhances the adaptability and flexibility of security policies. In the dynamic landscape of software development, where technologies and threats continually evolve, the ability to rapidly adapt and modify security policies is paramount. The SPMM, with its modular and scalable structure, facilitates this adaptability, enabling organizations to tailor their security measures to specific threats and changing environments. This adaptability is particularly crucial in sectors with varying security requirements, such as healthcare, finance, and government.

In summary, the introduced SPMM can represent and managing the aspects of security policies in software systems. Through its structured and modular approach, the SPMM contributes to enhancing the security posture of software systems, making them more resilient to evolving threats and adaptable to changing requirements.

(RQ2) - *How can the aspects of security policies in software systems be tailored towards the Concerns of Stakeholders?*

The process of tailoring security policies in software systems to address stakeholder concerns requires a nuanced approach that integrates several key elements of the SPCM. Firstly, understanding and identifying stakeholder concerns is paramount. These concerns, which could range from compliance with regulations to incident response and data protection, guide the development of security policies. Secondly, the construction of a *Security Policy Perspective* is critical. This perspective ensures that the security policy is focused on stakeholder-specific concerns, filtering the broader policy through a lens that highlights and addresses these concerns.

In addition to these elements, the incorporation of *Model Kinds* and *Views* is essential. These components provide a structured approach to represent the security policy in a manner that aligns with the stakeholder's understanding and requirements.

Furthermore, the introduction of *Automation Strategies* and *Automation Solutions*, as part of the SPACM, plays a significant role in operationalizing the security policy. This ensures that the security policy is not only theoretically sound but also practically applicable and responsive to the dynamic needs of stakeholders.

Ultimately, the key to successfully tailoring security policies lies in the careful alignment of these elements, ensuring that each stakeholder's specific concerns are thoroughly addressed and reflected in the SPMM.

(RQ3) - *In what sense can the aspects of security policies in software systems be automated and what are the challenges of automating them?*

Automating aspects of security policies in software systems involves translating the concepts modeled within a SPM into actionable formats. This translation can take various forms, such as scripts that execute responses to security breaches or partially generated source code and configurations. The output of automation is not always executable code or configurations. In certain cases, textual instructions may be included to guide stakeholders in manually implementing the modeled Security Policy.

The automation process is integrated with the SPMM, which involves a thorough examination of the model elements, their interrelationships, and how they come together within the SPMM. This approach ensures that the automation is consistent with the SPMM's structure, maintaining coherence and functionality. Thus, the challenge is not only technical but also involves ensuring that automated elements integrate and align with the SPMM.

The application examples illustrate the difficulties of automating a security policy because of the diverse concerns of stakeholders, despite a consistent underlying SPM. These examples demonstrate various types of automation. One approach utilizes a code generator to automate the process by generating code, which is then utilized to implement the modeled aspects of the security policy. Alternatively, another method provides instructions to help stakeholders complete tasks more accurately and in accordance with security policies.

In conclusion automating security policy means implementing various aspects of the security policy so that they can be executed without manual intervention. This varies by stakeholder. For example, for software developers, it might involve the automatic generation of source code, eliminating the need for manual integration of the security policy into the system. Other stakeholders may implement automation through the use of a CI/CD pipeline to enforce the SPM. It is critical to consider the concerns of each stakeholder, as they will determine which aspects of the security policy should be automated and what type of automation is required. Automation can be achieved in a number of ways, including code generation, CI/CD pipelines, or scripts. These can be used to ensure compliance with the security policy or to provide guidance to stakeholders on how to comply with the security policy in their interactions with the system.

9.3. Proposed Evaluation Process

This section proposes a process for evaluating our SPMM. Initially, the objectives and scope of the evaluation are established. This is followed by the design of the evaluation process. Our proposed evaluation methodology is based upon the evaluation process of Meta-Models introduced by Kläs et al. [Klä+10]

Objectives: Our approach aims to assess whether our SPMM achieves two primary objectives: Firstly, it should enable the creation of tailored SPMs using our SPCM, addressing stakeholder concerns effectively. Secondly, it should facilitate the modeling of automation for these tailored SPMs. Additionally, this approach evaluates if the resulting SPAMs remain in alignment with the overarching security policies.

Scope: The scope of the evaluation is confined to the concepts introduced in the SPMM. It does not extend to assessing the effectiveness of the security policy itself, or the efficacy of the resulting SPAM in achieving underlying security objectives. The evaluation focuses solely on whether the SPAM can adequately express the specified security policy. Furthermore, it does not assess the correctness of the concept model element entities, but rather evaluates their functionality within the SPMM.

9.3.1. Design

This section outlines the methodology for conducting an evaluation of our SPMM. Initially, it details the evaluation process, followed by an example catalog of questions for use in this process. Finally, it describes the approach for interpreting the results of the evaluation.

Evaluation Process: Figure 9.2 illustrates the evaluation process for a SPMM. The process begins with the security expert, who is responsible for the construction of the SPAM for a *Security Policy*, tailored towards a *Stakeholder*, using the construction process introduced in this thesis. The expertise of the security rxpert in both the *Security Policy* and the *Stakeholder*'s *Concerns* is crucial for construction an effective SPAM.

Once the SPAM is constructed, it is presented to the stakeholder, along with a catalog of questions. The stakeholder is tasked with answering the question from the catalog of questions.

Finally, the evaluation process ends in analyzing the stakeholder's responses to determine if the SPMM has successfully met the objectives of the evaluation.

The evaluation process, as described, should be conducted across a variety of security policies and stakeholders. This approach ensures a more comprehensive and robust evaluation, as it allows for the evaluation of the SPMM in diverse contexts and scenarios.

Catalog of Questions: To facilitate the comparison of responses provided by different stakeholders regarding various security policies, a structured approach is essential. We

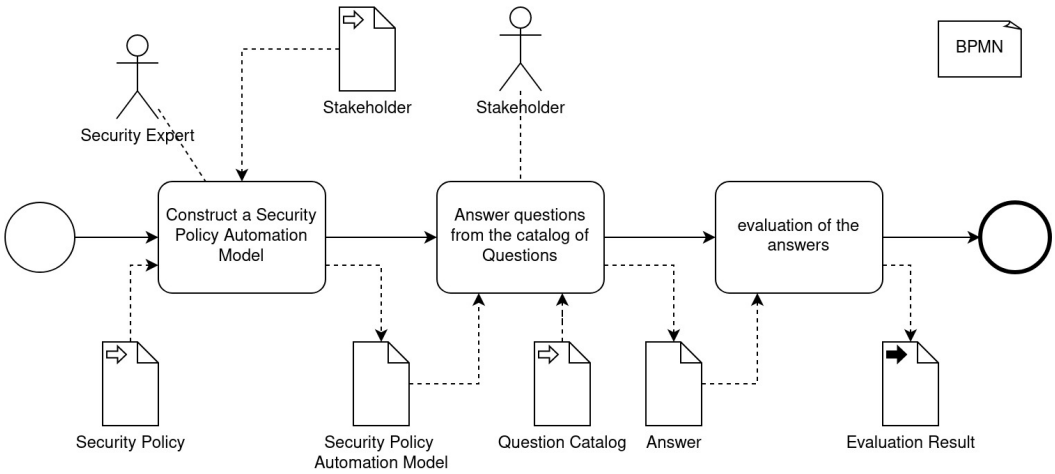

Figure 9.2.: Evaluation Process for a Security Policy for a Stakeholder

Question	Answer
Is the SPM understandable?	Yes or No
Are the modeled aspects of the Security Policy relevant for your Concerns?	Yes or No

Table 9.1.: Preliminary catalog of questions

recommend adopting a standardized questioning methodology to increase the comparability and quality of the responses. Table 9.1 presents a preliminary catalog of questions. The first column displays the questions, while the second column lists potential responses. Utilizing predefined answers enables the incorporation of numerous iterations in the proposed evaluation process.

As an alternative to the standardized questioning approach, conducting interviews with stakeholders is feasible. In this method, stakeholders respond to a predefined set of questions. Although this approach hinders the practicality and comparability of processing a large volume of responses, it can serve as an effective starting point for the evaluation process.

Interpretation: If the standardized questioning methodology is chosen, the interpretation focuses on quantitative analysis, enabling systematic comparison and aggregation of stakeholder responses. This approach lends itself to statistical evaluation, offering a clear, objective understanding of common trends and patterns across various responses. Conversely, if the interview approach is chosen, interpretation shifts towards qualitative analysis. Here, the emphasis is on the depth and context of individual responses, providing nuanced insights into stakeholders' perspectives. This method facilitates a more detailed, subjective interpretation of the underlying reasons and motivations behind each response.

10. Conclusion

Contents

To effectively understand and implement the automation of security policies in software systems, it is critical to model these policies and their automation comprehensively. This includes not only the technical aspects of modeling, but also addressing the concerns of all stakeholders involved in the software project. This thesis aims to establish a first approach to guide future work in this area. The following chapter provides a brief summary of the research methodology and the contributions of this thesis. The insights gained from our research have revealed numerous opportunities for further development and research. We provide a brief outlook on potential directions for future work that could extend the contributions and knowledge established in this thesis.

10.1. Summary

This thesis presents a novel SPMM designed for modeling security policies and automation of security policies tailored towards stakeholders. The SPMM addresses the gap in existing modeling approaches. Primarily, existing modeling approaches focus on technical aspects and do not adequately address the need to integrate security policies into all phases of the SDLC. Integration into all phases of the SDLC requires a modeling approach that is able to address the concerns of all stakeholders in the SDLC.

We start by conducting a lightweight SLR to establish foundational knowledge on security policy aspects, structure, tailoring to stakeholder concerns, and possibilities and challenges of automation. The results of the SLR highlighted the lack of a comprehensive modeling approach for security policies and their automation that addresses stakeholder concerns.

As a first step in bridging this gap, we have introduced the SPMM. The SPMM includes the SPCM and the SPACM. The SPCM supports the construction of the SPM. The GlsSPM is the model of a security policy that is tailored to stakeholder concerns. The SPACM supports the construction of the AM. The AM is a model of the automation of an SPM. Together, the SPM and AM form the SPAM.

In addition, we introduce a CP that guides the construction of the SPAM. The CP is

used to evaluate the soundness of the SPMM through the construction of three application examples (AE). The first AE evaluates the ability of the SPMM to be used to construct a SPAM. The second AE evaluates the ability of the SPMM to be used to construct an SPM that simultaneously addresses the concerns of multiple stakeholders. The third AE evaluates the ability of the SPMM to be used to construct multiple AMs for the same SPM, but where each AM addresses different stakeholders and has different automation goals.

The result of these application examples demonstrates the ability of the SPMM to construct SPAMs for different stakeholders without the need to construct separate SPAMs for each stakeholder. This approach significantly reduces the number of SPAMs required, providing practical advantages in the real world. In addition, the ability of SPMM to incorporate the concerns of different stakeholders into security policy automation helps to identify potential issues that are unique to specific stakeholders.

However, we acknowledge a limitation in our evaluation methodology: The AEs do not prove the applicability of SPMM to all types of security policies or stakeholders. In recognition of this, a process is proposed for future case studies to explore the broader applicability of the SPMM.

In conclusion, the SPMM shows promise in improving the understanding and implementation of security policies for various stakeholders. In addition, the process of security policy automation can potentially be improved by effectively addressing the specific concerns of stakeholders. Further research is needed to validate its universal applicability in different contexts.

10.2. Feature Work

The novel SPMM is a starting point for future work toward a syantmatic approch for modeling security policy automation that is tailored towards stakeholder concerns. In this section, we propose directions of feature research to expand upon the results of this thesis.

10.2.1. Case Studies

Future research directions could include conducting comprehensive case studies to evaluate the capabilities of the SPMM. The application examples used so far provide a basic understanding, but more extensive research is needed to fully evaluate the effectiveness of the SPMM in modeling security policy automation for different stakeholders.

These proposed case studies would explore a range of security policies and stakeholder perspectives to assess the adaptability and effectiveness of the SPMM. The aim is to determine whether the SPMM can facilitate the development of SPAM that are more understandable and tailored to the specific needs of different stakeholders compared to existing security policy models.

Utilizing the evaluation process described in previous chapters, these proposed studies would ensure a systematic and focused approach aimed at providing reliable and insightful

results. The focus would be on the practical application of the SPMM, specifically its effectiveness in modeling different security policies and automating them for a diverse group of stakeholders.

Findings from these proposed case studies would be critical in providing insights into the effectiveness of the SPMM and identifying areas where improvements may be needed. Such studies are essential for validating the broader applicability of the SPMM in different contexts, thereby contributing to its overarching goal of creating tailored SPMs that adequately address the needs and concerns of different stakeholders.

10.2.2. Additionally concept models

Building on the foundation of the SPACM developed, future work should explore the integration of additional concept models to more accurately reflect the different types of automation. While the current concept model provides a general framework, its scope is somewhat limited and may overlook the nuances of specific automation approaches. The development of concept models that address these specificities could significantly improve the quality of the overall SPMM by providing a more accurate representation of the automation landscape.

One direction for future research is to develop conceptual models that distinguish between technical and procedural automation. Technical automation, which focuses on aspects such as code generation and policy compliance verification, and procedural automation, which focuses on streamlining processes for non-technical stakeholders, could benefit from distinct, specialized models. This bifurcation would allow for a more granular review and implementation of security policies.

In addition, the relationship between the nature of security policies and their respective automation strategies warrants further investigation. Concept models that can adapt to the specific requirements of different types of security policies, whether they are architectural patterns or broader organizational rules, could provide more tailored automation solutions. This could involve segmenting security policies into aspects and designing automation strategies that are uniquely suited to each aspect, whether through generators, CI/CD pipelines, or application usage logs.

However, this approach is not without its challenges. Specialized conceptual models can result in a narrow focus, potentially limiting their applicability to specific domains. This trade-off between specificity and generalizability must be carefully considered. Nevertheless, the potential for increased expressiveness and precision in representing automation strategies within these models represents a promising avenue for future research.

The proposed extension of the SPACM, through the integration of specialized models, provides a path to a more nuanced and effective representation of security policy automation. This future work could significantly contribute to the research field by providing a more detailed and adaptable framework for security policy automation in software systems.

A. SLR Results

A.1. Distributed Middleware Enforcement of Event Flow Security Policy [Mig+10]

A.1.1. Abstract

Distributed, event-driven applications that process sensitive user data and involve multiple organisational domains must comply with complex security requirements. Ideally, developers want to express security policy for such applications in data-centric terms, controlling the flow of information throughout the system. Current middleware does not support the specification of such end-to-end security policy and lacks uniform mechanisms for enforcement. We describe DEFCON-POLICY, a middleware that enforces security policy in multi-domain, event-driven applications. Event flow policy is expressed in a high--level language that specifies permitted flows between distributed software components. The middleware limits the interaction of components based on the policy and the data that components have observed. It achieves this by labelling data and assigning privileges to components. We evaluate DEFCON-POLICY in a realistic medical scenario and demonstrate that it can provide global security guarantees without burdening application developers.

A.2. Selection of regression system tests for security policy evolution [Hwa+12]

A.2.1. Abstract

As security requirements of software often change, developers may modify security policies such as access control policies (policies in short) according to evolving requirements. To increase confidence that the modification of policies is correct, developers conduct regression testing. However, rerunning all of existing system test cases could be costly and time-consuming. To address this issue, we develop a regression-test-selection approach, which selects every system test case that may reveal regression faults caused by policy changes. Our evaluation results show that our test-selection approach reduces a substantial number of system test cases efficiently. Copyright 2012 ACM.

Web of Science	(TI=(security) AND TI=(polic*)) AND (ALL=(model*) OR ALL=(component*) OR ALL=(propert*)) AND AB=(software)	44
IEEExplore	("Document Title":"security" AND "Document Title":"polic*") AND ("Full Text .AND. Metadata":model* OR "Full Text .AND. Metadata":propert* OR "Full Text .AND. Metadata":component*) AND "Abstract":software	87
ACM Digital Library	Title:(security AND polic*) AND AllField:(model* component* propert*) AND Abstract:(software)	34
Science Direct	terms: model models component components property properties Title, abstract or author-specified keywords "security policy" software	59
Scopus	TITLE ("security polic*") AND (ALL (model*) OR ALL (component*) OR ALL (propert*)) AND ABS (software) AND (LIMIT-TO (SUBJAREA , "COMP"))	102

Table A.1.: SLR Search Queries

A.3. A Flexible Architecture for Systematic Implementation of SoC Security Policies [BBR15]

A.3.1. Abstract

Modern SoC designs incorporate several security policies to protect sensitive assets from unauthorized access. The policies affect multiple design blocks, and may involve subtle interactions between hardware, firmware, and software. This makes it difficult for SoC designers to implement these policies, and system validators to ensure adherence. Associated problems include complexity in upgrading these policies, IP reuse for systems targeted for markets with differing security requirement, and consequent increase in design time and time-to-market. In this paper, we address this important problem by developing a generic, flexible architectural framework for implementing arbitrary security policies in SoC designs. Our architecture has several distinctive features: (1) it relies on a dedicated, centralized, firmware-upgradable plug-and-play IP block that can implement diverse security policies; (2) it interfaces with individual IP blocks through their "security wrapper", which exploits and extends test/debug wrappers; (3) it implements a security policy as firmware code following existing security policy languages; (4) it can implement any security policy as long as relevant observable and controllable signals from the constituent IPs are accessible through the security wrappers; and (5) it realizes a low-overhead communication link between security wrappers of IP blocks and the centralized, dedicated controller. The approach builds on and extends the recent work on developing a centralized infrastructure IP for SoC security, referred to as IIPS, that interface with IP blocks using their boundary scan based wrappers. While this architecture is generic and independent of security policy types, we provide case studies with several common policies to show the flexibility and extendibility of the architecture. We also evaluate its viability in terms of overhead in area and power.

A.4. How Good Is a Security Policy against Real Breaches? A HIPAA Case Study [Kaf+17]

A.4.1. Abstract

Policy design is an important part of software development. As security breaches increase in variety, designing a security policy that addresses all potential breaches becomes a nontrivial task. A complete security policy would specify rules to prevent breaches. Systematically determining which, if any, policy clause has been violated by a reported breach is a means for identifying gaps in a policy. Our research goal is to help analysts measure the gaps between security policies and reported breaches by developing a systematic process based on semantic reasoning. We propose SEMAVER, a framework for determining coverage of breaches by policies via comparison of individual policy clauses and breach descriptions. We represent a security policy as a set of norms. Norms (commitments, authorizations, and prohibitions) describe expected behaviors of users,

and formalize who is accountable to whom and for what. A breach corresponds to a norm violation. We develop a semantic similarity metric for pairwise comparison between the norm that represents a policy clause and the norm that has been violated by a reported breach. We use the US Health Insurance Portability and Accountability Act (HIPAA) as a case study. Our investigation of a subset of the breaches reported by the US Department of Health and Human Services (HHS) reveals the gaps between HIPAA and reported breaches, leading to a coverage of 65

A.5. When role models have flaws Static validation of enterprise security policies [Pis+07a]

A.5.1. Abstract

Modern multiuser software systems have adopted Role-Based Access Control (RBAC) for authorization management. This paper presents a formal model for RBAC policy validation and a static-analysis model for RBAC systems that can be used to (i) identify the roles required by users to execute an enterprise application, (ii) detect potential inconsistencies caused by principal-delegation policies, which are used to override a user's role assignment, (iii) report if the roles assigned to a user by a given policy are redundant or insufficient, and (iv) report vulnerabilities that can result from unchecked intra-component accesses. The algorithms described in this paper have been implemented as part of IBM's Enterprise Security Policy Evaluator (ESPE) tool. Experimental results show that the tool found numerous policy flaws, including ten previously unknown flaws from two production-level applications, with no false-positive reports. © 2007 IEEE.

A.6. Modeling and verification of ATM security policies with SecBPMN [SG14]

A.6.1. Abstract

High Performance Computing (HPC) techniques are essential in complex systems such as Socio-Technical Systems (STSs), where humans and organizations are elements of the same system along with technical infrastructures and hardware/software components. For example, several HPC approaches have been successfully applied to support and facilitate distribution or aggregation of computation power among independent and atomic components (e.g., smart meters to solve and/or simulate complex models). However, HPC techniques have to be studied and developed without underestimating the problem of security that, given the interaction-centric nature of STSs, has to be considered not only from the single component perspective but for the system as a whole. In our previous work, we have proposed SecBPMN, a framework to support the design of secure STSs. It is used to model the interaction design and security policies of a STS and it supports their verification through a querying engine. In this paper, we describe how SecBPMN has been successfully used for the study of security in an Air Traffic Management (ATM)

system, and we show how it can result also an efficient support when of HPC techniques when applied in complex and heterogeneous environments. © 2014 IEEE.

A.7. Research and application of XACML-based fine-grained security policy for distributed system [SY13]

A.7.1. Abstract

In distributed system development security is a major design criteria. However, the present software system modeling does not deal with security. System security policies and mechanisms often become the supplement at the end of system development. In this paper, a software development process is considered two common stages which are functional requirement analysis stage and system design stage. We extend the UML notation to model the security requirements which is generated from the function modeling process. The access control UML models are finally used to generate access control policies which can be described by XACML and deployed into the enforcement infrastructure. Finally, the decisions for users' requests will be made through Sun's XACML Implementation. © 2013 IEEE.

A.8. Enforcing Policy-Based Security Models for Embedded SoCs within the Internet of Things [Hag+18]

A.8.1. Abstract

Within complex IoT ecosystems and network structures, hard to find vulnerabilities have potential to cause significant disruption and damage. In addition, device tampering and re-purposing can threaten business models of service providers. The vulnerability surface area of the ecosystem ranges across the entire system architecture, from the cloud to the IoT device. These can be introduced at any stage of the device life-cycle, including design, programming, manufacturing, integration, operation and maintenance of the device. While threat modelling during the design phase can alleviate some potential vulnerabilities, it is more difficult or even impossible to mitigate problems for devices already in the market. A policy-based device security model is proposed as an approach, that can be enforced using hardware and software security architectures. This paper reflects on existing literature on threat modelling and how derived security models can influence the design phase. This contribution proposes that by using the threat modelling to define specific use case security policies within the security model, OEMs will be able to tailor their solution to conform to the user's security requirements. Platform vendors, on the other hand, will have reduced design costs as they can offer generic solutions for differing levels of criticality. An example scenario is provided using an industrial PLC as the attack target. While threat modelling can establish countermeasures for both the design process and policy defining, the policy can be introduced quickly, whereas the design method approach requires extensive modification to the system firmware.

A.9. Formalizing the Relationship between Security Policies and Objectives in Software Architectures [Rou+23]

A.9.1. Abstract

It is difficult to ensure the relationship between the selected security solutions and the security concerns in software architecture design. Assuring this analysis manually is labor intensive, expertise dependent and error prone. As a result, there is an increasing need for the development of reusable security solutions to aid in the early stages of secure system development. This paper proposes an approach for designing secure component-based software architectures using analysis around two main aspects: (1) the use of formal methods to formally verify and capture the satisfaction relationship between policies (security solutions) and the targeted security objectives (problems) and, (2) the definitions of these policies and security objectives as properties and provided as formal model libraries to support their reuse. To validate our work, we investigate a representative confidentiality security objective extracted from the CIA (confidentiality, integrity, and authenticity) classification and its relationship with a related policy in the context of secure component-based software architecture development. The proposed model can assist system designers in reworking their designs in order to satisfy the identified security objectives. In this way, our approach can help in the development of dependable software systems. Finally, we propose an MDE-based tool to support the approach and automate part of the analysis using reusable formal models. © 2023 IEEE.

A.10. Required Policies and Properties of the Security Engine of an SoC [MRF21]

A.10.1. Abstract

With the increasing complexity of system-on-chip (SoC) designs, security has become a vital requirement. The confidentiality and integrity of critical information, access controls as well as chip authentication at both software and hardware levels should be guaranteed for SoCs. A secure and trusted component is necessary to provide those required security and trust mechanisms in SoCs. The goal of this component is to provide support for security-critical operations and functionalities like provisioning and protection of assets, watermark generation, intellectual property (IP) unlocking, as well as providing isolation at the hardware and software levels. In this paper, we provide a comprehensive overview of the requirements and components for the design of a root-of-trust (RoT) termed as Security Engine that protects against various attacks at the manufacturing floor and during in-field operations while providing security-critical functionalities and features. In addition to that, we identify several critical protocols and security policies for RoT. Policies ensure secure operations and safe transfer of assets while maintaining confidentiality and integrity. Policies are in the form of access control, data integrity and retention, encryption, and asset management for a Security Engine. Similarly, we

identify several critical functionalities and protocols of the SoC development like secure boot, self-test, provisioning protocols, security IPs, watermark generation, secure debug, etc., and give a conceivable solution for every one of them with the assistance of the proposed Security Engine while making not many presumptions. Policies and protocols can be implemented in hardware and software with minimum overhead. They can also be checked and enforced by integrating them into the firmware code of the RoT processor. Moreover, this paper will define different types of security policies like access control, data integrity and retention, encryption, and asset management policy for a Security Engine, which is the hub of security operations in an SoC.

A.11. A policy-based security model for Web system [XM03]

A.11.1. Abstract

The browser/server-based software development model is a popular way to for the application on the Internet. Thus, there are more and more software systems based on this framework, such as electronic commerce, electronic government, management information system. However, the security becomes the key factor to the application system on Web. Security policy is the bedrock to implement system security in variable circumstances. In this paper, we try to create a method to measure security degree by some tools such as fuzzy assessment and graph theory, then stipulate for security policy. This method provides a new way to analyse system security. This policy-based security model for Web application improves the adaptability and evolvement of secure system.

A.12. FSM Modeling of Testing Security Policies for MapReduce Frameworks [HAC19]

A.12.1. Abstract

Nowadays, MapReduce becomes one of the most common computational paradigms that provide an efficient parallel processing of large-scale data especially in public clouds in order to enable users to process data without considering physical infrastructures and software installations. However, the deployment of MapReduce in such public environment needs to deal with many security threats attacks during networks communication. In this paper, we present a novel approach to test Access Control List (ACL) policies in MapReduce framework. The proposed system uses these ACLs to control which users can submit to certain queues as well as which users can administer a queue for example. To this end, we used the FSM formalism to write our system specification that takes into account these policies expressed in XACML language.

A.13. Trust-adapted enforcement of security policies in distributed component-structured applications [HK01]

A.13.1. Abstract

Software component technology on the one hand supports the cost-effective development of specialized applications. On the other hand, however, it introduces special security problems. Some major problems can be solved by the automated run-time enforcement of security policies. Each component is controlled by a wrapper which monitors the component's behavior and checks its compliance with the security behavior constraints of the component's employment contract. Since control functions and wrappers can cause substantial overhead, we introduce trust-adapted control functions where the intensity of monitoring and behavior checks depends on the level of trust, the component, its hosting environment, and its vendor have currently in the eyes of the application administration. We report on wrappers and a trust information service, shortly outline the embedding security model and architecture, and describe a Java Bean based experimental implementation.

A.14. SDN testbed for validation of cross-layer data-centric security policies [Wro+17]

A.14.1. Abstract

Software-defined networks offer a promising framework for the implementation of cross-layer data-centric security policies in military systems. An important aspect of the design process for such advanced security solutions is the thorough experimental assessment and validation of proposed technical concepts prior to their deployment in operational military systems. In this paper, we describe an OpenFlow-based testbed, which was developed with a specific focus on validation of SDN security mechanisms - including both the mechanisms for protecting the software-defined network layer and the cross-layer enforcement of higher level policies, such as data-centric security policies. We also present initial experimentation results obtained using the testbed, which confirm its ability to validate simulation and analytic predictions. Our objective is to provide a sufficiently detailed description of the configuration used in our testbed so that it can be easily re-plicated and re-used by other security researchers in their experiments.

A.15. An Automated Validation Method for Security Policies the firewall case [AE08b]

A.15.1. Abstract

Research in computer security issues has recently addressed the development of Security Policy specification languages. It has however omitted the need of formal validation. In

this paper we try to remedy to this drawback by the proposition of an automated tool for security policies. Because we have found several similarities between security policies and software engineering, our approach is strongly inspired from the reasoning followed in the software engineering. First, it brings out a model inspired by Promela to enable the validation task. Secondly, it proposes a 3-step validation process that deals with consistency, completeness and presenation of safety and liveness properties.

A.16. Embedding Model-Based Security Policies in Software Development [NC16]

A.16.1. Abstract

Security in software applications is frequently an afterthought. Even if developers are aware of security policies and software vulnerabilities, they possess little knowledge of how to implement security polices while developing applications. In addition, the lack of support for tools and security automation makes it more challenging to incorporate security policies. In this paper we have proposed a security policy enforcement mechanism to incorporate security policies for data fields in transactions of software application during its development phase. The objective is to facilitate developers implementing security policies easily. The extensibility of our approach gives the flexibility to accommodate different security policy schemas and to implement various security policies on sensitive data. With the simplicity of mapping data fields of business structures with security policy definitions, our approach provides the programmers, business domain experts and security experts a collaborative process to define and incorporate security policies in software.

A.17. Policy-based security channels for protecting network communication in mobile cloud computing [IKC11]

A.17.1. Abstract

In this paper we present a set of policy-driven security protocols for ensuring the confidentiality and integrity of enterprise data in mobile cloud computing environments. The proposed protocols leverage trusted authority entities and the “elastic” virtualized nature of the cloud computing model to provide energy-efficient key management mechanisms and policy-driven data protection techniques that support the secure interaction of the mobile client with an assortment of cloud software and storage services. The main contribution lies in: (1) Offloading the intensive asymmetric key agreement mechanisms from the mobile client and delegating them to resource-lucrative trusted authority sites. This is achieved by aggregating the security associations, required to agree on symmetric keys between the client and the cloud services, in a single security association between the client and the trusted authority. The aggregation concept results in major energy savings especially when the client consumes a relatively large set of services as is the case in

cloud computing today. (2) Designing a customizable policy-based security architecture that considers the sensitivity of cloud data to provide multi-level and fine-grained data protection methodologies that suit the energy-limited mobile devices and the low-bandwidth wireless networks characterizing current mobile cloud computing models. The system is implemented in a real cloud computing environment and the savings in terms of energy consumption and execution time are analyzed.

A.18. Verify consistency between security policy and firewall policy with answer set programming [LD08]

A.18.1. Abstract

Firewalls are core elements in network security, the effectiveness of firewall security is dependent on configuring firewall policy correctly. Firewall policy is a lower-level policy which describes how firewall actually implements security policy. Security policy is a higher-level policy which defines the access that will be permitted or denied from the trusted network. Compare with software engineer, security policy is a design, firewall policy is a set of codes. It is useful to discover inconsistency between security policy and firewall Policy. In this paper, we present a method of verifying consistency between security policy and firewall policy, which applies the idea of model checking. First of all, two policies and the consistency are represented with logic programs. Then the verification is applied by testing whether the logic formula of the consistency is satisfied in the semantics of the logic programs. Furthermore, We prove that the method has an unique answer which can be computed in polynomial time. © 2008 IEEE.

A.19. Analysis of Policy-Based Security Management System in Software-Defined Networks [Soo+19]

A.19.1. Abstract

In software-defined networks, policy-based security management or architecture (PbSA) is an ideal way to dynamically control the network. We observe that on the one hand, this enables security capabilities intelligently and enhance fine-grained control over end user behavior. But, on the other hand, dynamic variations in network, rapid increases in security attacks, geographical distribution of nodes, complex heterogeneous networks, and so on have serious effects on the performance of PbSAs. These affect the flow specific quality of service requirements with further degradation of the performance of the security context. Hence, in this letter, PbSA's performance is evaluated. The key factors including a number of rules, rule-table size, position of rules, flow arrival rate, and CPU utilization are examined, and found to have considerable impact on the performance of PbSAs.

A.20. Managing security policy in a large distributed Web services environment [CCH03]

A.20.1. Abstract

Effectively managing security policies in a large distributed Web Services environment is the key to secure e-business transactions. Security policy must ensure the end-to-end agreement for many-to-many interoperation; ensure the versioning interoperability and privacy of collaborating partners; and ensure the dynamic establishment of security policies because any statically defined security policy tends to be unsecured after a certain period of time. The traditional security policy configuration mechanisms, either the local configuration mechanism or the centralized configuration mechanism, cannot fully meet the above requirements. In this paper we describe a solution for managing security policies in a collaborative Web Services environment. This solution is based on ebXML CPP/CPA model and uses Interoperability Contract Document (ICD). It allows the collaboration parties to establish security policy dynamically for each individual interoperation; makes the selected policy confidential; and addresses the software, message, and policy versioning and interoperability issues. Our experience reveals the advantages of this approach over others.

A.21. A model for the analysis of security policies in service function chains [Dur+17]

A.21.1. Abstract

Two emerging architectural paradigms, i.e., Software Defined Networking (SDN) and Network Function Virtualization (NFV), enable the deployment and management of Service Function Chains (SFCs). A SFC is an ordered sequence of abstract Service Functions (SFs), e.g., firewalls, VPN-gateways, traffic monitors, that packets have to traverse in the route from source to destination. While this appealing solution offers significant advantages in terms of flexibility, it also introduces new challenges such as the correct configuration and ordering of SFs in the chain to satisfy overall security requirements. This paper presents a formal model conceived to enable the verification of correct policy enforcements in SFCs. Software tools based on the model can then be designed to cope with unwanted network behaviors (e.g., security flaws) deriving from incorrect interactions of SFs in the same SFC. © 2017 IEEE.

A.22. TechNETium Atomic Predicates and Model Driven Development to Verify Security Network Policies [Ber+20]

A.22.1. Abstract

Fifth-generation (5G) networks will deliver unprecedented levels of quality of service for online gaming and multimedia-rich social interaction, providing virtual environments optimized for vertical applications through innovative approaches to physical resource management. These techniques must consider security aspects in all phases and at every layer. Trusted communications between individuals and reliable platforms running services for social good depend on the resiliency to network-level attacks such as hijacking and denial-of-service. The verification of topological properties represents a well-suited approach to address these issues in a 5G environment. This paper illustrates moves from formal methods existing in literature, namely atomic predicates (AP) and header space analysis (HSA). It describes a method of integrating AP in Software Defined Network architectures, achieving the same expressive power as HSA without its performance hit, to make topology verification viable for real-time security applications.

A.23. Analyzing RBAC Security Policy of Implementation Using AST [PTN09]

A.23.1. Abstract

Security policy is a critical property in software applications which require high levels of safety and security. It has to be clearly specified in requirement documents and its implementation must be conformed to the specification. In this paper, we propose an approach to check if the implementation is in accordance with its security policy specification. We use the Abstract Syntax Tree (AST), another manner of expressing the program, to analyze the source code and specify user permission policy in software systems by Role-Based Access Control (RBAC).

A.24. A flexible architecture for security policy enforcement [MP03]

A.24.1. Abstract

Significant progress has been made on the design of security policy representations for complex communication systems. A significant problem however remains of how to design software architectures that enforce ever-changing security policy requirements efficiently. This research summary describes the security policy enforcement architecture of the Antigone 2.0 group communication system. The architecture is designed to be flexible: new security mechanism modules are added as needed to support emerging

policy requirements. Such mechanisms regulate the processing of system and network events as directed by the policy and enforce fine-grained control over sensitive data. A software bus is used coordinate the delivery of these events to mechanisms within each process. We summarize an analysis of the performance of the architecture and show that the overheads are modest for typical environments.

A.25. A model for specification and validation of security policies in communication networks The firewall case [AE08a]

A.25.1. Abstract

A security policy constitutes one of the major actors in the protection of communication networks. For this, and in order to manage the access grants in accordance with the security constraints, a security policy has to be validated before its deployment. Unfortunately, in the literature, there is no well established validation mechanisms ensuring the well founded of such security policies. This paper proposes a validation framework for security policies where: (1) executable specifications are used to build an 'Executable Security Policy', (2) a validation model is proposed to support the validation activity, and (3) a validation of the executable security policy is performed. The main contributions provided by this paper concerns the adaptation of some concepts and mechanisms traditionally used in software engineering for validation aims, such as specification, executable specification or reachability graph. All the definitions made in this paper have been proposed in accordance with the firewall case. © 2008 IEEE.

A.26. Using security policies in a network securing process [AF11]

A.26.1. Abstract

A security policy constitutes one of the major actors in the protection of communication networks but can be one of their drawbacks too. This can be the case if it is inadequate to the security requirements for example. For this, a security policy has to be checked before its real deployment. In this paper, we propose three checking activities each of which is adapted to a given phase of the policy deployment process. This activities deal with the SP validation, the SP testing and the multi-SP conflict management. Our techniques are inspired by the well established techniques of the software engineering for which we have found some similarities with the security domain. © 2011 IEEE.

A.27. Towards policy unification for enterprise network security [YZG17]

A.27.1. Abstract

The task of securing the enterprise network currently involves the use of several specialized devices (i.e. 'middleboxes') to provide specific functions in order to satisfy a set of high-level defined objectives. With only a loose common goal of securing the network, these employed devices could not be more distinct in functionality - from network access-control, vulnerability assessment, to intrusion detection and prevention systems, to firewalls. Network operators are therefore faced with the immense challenge of having to manage hundreds to thousands of configuration lines across these devices, in addition to providing the necessary coordination between these disparate functions, in order to effectively secure the network. Towards the aim of unifying policy enforcement across network security functions, we present an architecture that is based on Software-Defined Networking (SDN). Our solution extends the IEEE 802.1X framework and abstracts network endpoint connectivity context, thereby allowing for cross-functional policy composition and enforcement. We present a proof-of-concept prototype and performed some experimental evaluation, as well as outlining our future research directions.

A.28. Policy Based Security Middleware as a Service [Chr+14]

A.28.1. Abstract

Cloud computing is amongst the major topics researched in Computer Science. Cloud computing security is at the heart of the topic. The reason behind this is that, as a technology, cloud computing is a great alternative that allows companies to outsource the management of their computing resources. However, due to the security issues, the cloud technology is not reaching its full potential. In this paper we present a system that allows the enforcement of privacy, accountability, integrity, and availability in each one of cloud computing levels using software policies. We present and describe the architecture of the Policy-Based Security Middleware as a Service.

A.29. Modeling security-enhanced Linux policy specifications for analysis [ALP03b]

A.29.1. Abstract

Security-Enhanced (SE) Linux is a modification of Linux initially released by NSA in January 2001 that provides a language for specifying Linux security policies and, as in the Flask architecture, a security server for enforcing policies defined in the language. To determine whether user requests to the operating system should be granted, the security server refers to an internal form of the policy compiled from the policy specification. Since the most convenient description of the policy for user understanding is its source

specification in the policy language, it is natural for users to expect to be able to analyze the properties of the policy from this source specification. However, though specifications in the SE Linux policy language avoid implementation details, the policy language is very low-level, making the high level properties of a policy difficult to deduce by inspection. For this reason, tools to help users with the analysis are necessary. The goal of the NRL project on analyzing SE Linux security policies is to first use mechanized support to analyze the specification of an example policy, and then to customize this support for use by practitioners in the open source software community. This paper summarizes how we have modeled an example security policy in the analysis tool TAME, the kinds of analysis we can support, and prototype mechanical support to enable others to model example security policies in TAME. (For an extended version of this paper, see [5].)

A.30. Enforcing multilevel security policies in database-defined networks using row-level security [AA19]

A.30.1. Abstract

Despite the wide of range of research and technologies that deal with the problem of routing in computer networks, there remains a gap between the level of network hardware administration and the level of business requirements and constraints. Not much has been accomplished in literature in order to have a direct enforcement of such requirements on the network. This paper presents a new solution in specifying and directly enforcing security policies to control the routing configuration in a software-defined network by using Row-Level Security checks which enable fine-grained security policies on individual rows in database tables. We show, as a first step, how a specific class of such policies, namely multilevel security policies, can be enforced on a database-defined network, which presents an abstraction of a network's configuration as a set of database tables. We show that such policies can be used to control the flow of data in the network either in an upward or downward manner. © 2019 IEEE.

A.31. Evaluating the Use of Security Tags in Security Policy Enforcement Mechanisms [Alv+15]

A.31.1. Abstract

Security tagging schemes are known as promising mechanisms for providing security features in computer systems. Tags carry information about the tagged data throughout the system to be used in access control and other security mechanisms. This paper discusses several different uses of security tags related to different security policies, highlighting appropriate uses of the tags. The evaluation of the use of tags is presented in the summary of three security tagging application domains. One domain, using hardware-based tagging to prevent high-level attacks, was not found to be feasible. A

project to use hardware-based tagging for OS security enhancement and a project that uses software-based tagging for multi-level secure document management were successful.

A.32. A novel approach for integrating security policy enforcement with dynamic network virtualization [Bas+15]

A.32.1. Abstract

Network function virtualization (NFV) is a new networking paradigm that virtualizes single network functions. NFV introduces several advantages compared to classical approaches, such as the dynamic provisioning of functionality or the implementation of scalable and reliable services (e.g., adding a new instance to support demands). NFV also allows the deployment of security controls, like firewalls or VPN gateways, as virtualized network functions. However, currently there is not an automatic way to select the security functions to enable and to configure the selected ones according to a set of user's security requirements. This paper presents a first approach towards the integration of network and security policy management into the NFV framework. By adding to the NFV architecture a new software component, the Policy Manager, we provide NFV with an easy and effective way for users to specify their security requirements and a process that hides all the details of the correct deployment and configuration of security functions. To perform its tasks, the Policy Manager uses policy refinement techniques. © 2015 IEEE.

A.33. Analyzing security-enhanced Linux policy specifications [ALP03a]

A.33.1. Abstract

NSA's Security-Enhanced (SE) Linux enhances Linux by providing a specification language for security policies and a Flask-like architecture with, a security server for enforcing policies defined in the language. It is natural for users to expect to be able to analyze the properties of a policy from its specification in the policy language. But this language is very low level, making the high level properties of a policy difficult to deduce by inspection. For this reason, tools to help users with the analysis are necessary. The NRL project on analyzing SE Linux policies aims first to use mechanized support to analyze an example policy specification and then to customize this support for use by practitioners in the open source software community. This paper describes how we model policies in the analysis tool TAME, the kinds of analysis we can support, and prototype mechanical support to enable others to model their policies in TAME. The paper concludes with some general observations on desirable properties for a policy language.

A.34. Compile-time enforcement of dynamic security policies [Eye+08]

A.34.1. Abstract

Dynamic separation of duties, delegation and other dynamic security constraints require the state of the security system to be managed explicitly at run-time in software. The majority of this software is still programmed directly by humans, and is thus susceptible to errors that will impact the overall functionality and security of the system. In this paper we demonstrate a technique for statically checking properties of the software that manages dynamic security policies. We base our work on Kilim, a shared-nothing, message-passing Java framework that provides a faster safer alternative to the dominant shared-memory and locking paradigm. We demonstrate that Kilim's static, compile-time verification of type linearity can also effect validation of aspects of dynamic security systems. We describe our initial steps toward the use of Kilim to support active, distributed security infrastructure.

A.35. OpenSec Policy-Based Security Using Software-Defined Networking [LR16]

A.35.1. Abstract

As the popularity of software-defined networks (SDN) and OpenFlow increases, policy-driven network management has received more attention. Manual configuration of multiple devices is being replaced by an automated approach where a software-based, network-aware controller handles the configuration of all network devices. Software applications running on top of the network controller provide an abstraction of the topology and facilitate the task of operating the network. We propose OpenSec, an OpenFlow-based security framework that allows a network security operator to create and implement security policies written in human-readable language. Using OpenSec, the user can describe a flow in terms of OpenFlow matching fields, define which security services must be applied to that flow (deep packet inspection, intrusion detection, spam detection, etc.) and specify security levels that define how OpenSec reacts if malicious traffic is detected. In this paper, we first provide a more detailed explanation of how OpenSec converts security policies into a series of OpenFlow messages needed to implement such a policy. Second, we describe how the framework automatically reacts to security alerts as specified by the policies. Third, we perform additional experiments on the GENI testbed to evaluate the scalability of the proposed framework using existing datasets of campus networks. Our results show that up to 95

A.36. A software architecture for automatic security policy enforcement in distributed systems [HBM07]

A.36.1. Abstract

Policies, which are widely deployed in networking services (e.g., management, QoS, mobility, etc.), are a promising solution for securing wide distributed systems. However, the adoption of a policy-based approach for security requires an appropriate policy specification and enforcement tools. In fact, A long-standing problem in distributed systems security is how to specify and enforce correctly security policies. In this paper, we mainly focus on how to systematically specify correct policies instead of manually configuring them and how to automatically enforce security policies in distributed systems. A software engineering approach is presented to overcome these issues. This approach is based on design and developpement of a software architecture to automating definition and enforcing policies. © 2007 IEEE.

A.37. A Policy-Based Security Architecture for Software-Defined Networks [Var+19]

A.37.1. Abstract

As networks expand in size and complexity, they pose greater administrative and management challenges. Software-defined networks (SDNs) offer a promising approach to meeting some of these challenges. In this paper, we propose a policy-driven security architecture for securing end-to-end services across multiple SDN domains. We develop a language-based approach to design security policies that are relevant for securing SDN services and communications. We describe the policy language and its use in specifying security policies to control the flow of information in a multi-domain SDN. We demonstrate the specification of fine-grained security policies based on a variety of attributes, such as parameters associated with users and devices/switches, context information, such as location and routing information, and services accessed in SDN as well as security attributes associated with the switches and controllers in different domains. An important feature of our architecture is its ability to specify path- and flow-based security policies that are significant for securing end-to-end services in SDNs. We describe the design and the implementation of our proposed policy-based security architecture and demonstrate its use in scenarios involving both intra- and inter-domain communications with multiple SDN controllers. We analyze the performance characteristics of our architecture as well as discuss how our architecture is able to counteract various security attacks. The dynamic security policy-based approach and the distribution of corresponding security capabilities intelligently as a service layer that enables flow-based security enforcement and protection of multitude of network devices against attacks are important contributions of this paper.

A.38. Brew A Security Policy Analysis Framework for Distributed SDN-Based Cloud Environments [Pis+19]

A.38.1. Abstract

The ease of programmability in Software-Defined Networking (SDN) makes it a great platform implementation of various initiatives that involve application deployment, dynamic topology changes, and decentralized network management in a multi-tenant data center environment. However, implementing security solutions in such an environment is fraught with policy conflicts and consistency issues with the hardness of this problem being affected by the distribution scheme for the SDN controllers. In this paper we present Brew, a security policy analysis framework implemented on an OpenDaylight SDN controller, that has comprehensive conflict detection and resolution modules to ensure that no two flow rules in a distributed SDN-based cloud environment have conflicts at any layer; thereby assuring consistent conflict-free security policy implementation and preventing information leakage. We present techniques for global prioritization of flow rules in a decentralized environment, extend firewall rule conflict classification from a traditional environment to SDN flow rule conflicts by recognizing and classifying conflicts stemming from cross-layer conflicts and provide strategies for unassisted resolution of these conflicts. Alternately, if administrator input is desired to resolve conflicts, a novel visualization scheme is implemented to help the administrators view the conflicts graphically. We demonstrate the correctness, feasibility and scalability of our framework through a proof-of-concept prototype. © 2004-2012 IEEE.

A.39. Intuitive Security Policy Configuration in Mobile Devices Using Context Profiling [Gup+12]

A.39.1. Abstract

Configuring access control policies in mobile devices can be quite tedious and unintuitive for users. Software designers attempt to address this problem by setting up default policy configurations. But such global defaults may not be sensible for all users. Modern smart phones are capable of sensing a variety of information about the surrounding environment like Bluetooth devices, WiFi access points, temperature, ambient light, sound and location coordinates. We conjecture that profiling this type of contextual information can be used to infer the familiarity and safety of a context and aid in access control decisions. We propose a context profiling framework and describe device locking as an example application where the locking timeout and unlocking method are dynamically decided based on the perceived safety of current context. We report on using datasets from a large scale smart phone data collection campaign to select parameters for the context profiling framework. We also describe a prototype implementation on a smart phone platform. More generally, we hope that our example design and implementation spurs further research on the notion of using context profiling towards automating security

policy decisions and identify other applications.

A.40. Extending the Java Virtual Machine to Enforce Fine-Grained Security Policies in Mobile Devices [IDC07]

A.40.1. Abstract

The growth of the applications and services market for mobile devices is currently slowed down by the lack of a flexible and reliable security infrastructure. The development and adoption of a new generation of mobile applications depends on the end user's ability to finely manage system security and control application's behavior. The virtual execution environment for mobile software and services should support the security needs of users and applications. This paper proposes an extension to the security architecture of the java virtual machine for mobile systems, to support fine-grained policy specification and run-time enforcement. Access control decisions are based on system state, application and system history data, as well as request specific parameters. The prototype implementation is running on desktops, as emulator, and on mobile devices, proving the high level of flexibility and security, with excellent performance provided by the extended architecture.

A.41. Security policy checking in distributed SDN based clouds [PCH16]

A.41.1. Abstract

Separation of network control from devices in Software Defined Network (SDN) allows for centralized implementation and management of security policies in a cloud computing environment. The ease of programmability also makes SDN a great platform implementation of various initiatives that involve application deployment, dynamic topology changes, and decentralized network management in a multi-tenant data center environment. Dynamic change of network topology, or host reconfiguration in such networks might require corresponding changes to the flow rules in the SDN based cloud environment. Verifying adherence of these new flow policies in the environment to the organizational security policies and ensuring a conflict free environment is especially challenging. In this paper, we extend the work on rule conflicts from a traditional environment to an SDN environment, introducing a new classification to describe conflicts stemming from cross-layer conflicts. Our framework ensures that in any SDN based cloud, flow rules do not have conflicts at any layer; thereby ensuring that changes to the environment do not lead to unintended consequences. We demonstrate the correctness, feasibility and scalability of our framework through a proof-of-concept prototype.

A.42. Security policy enforcement in the OSGi framework using aspect-oriented programming [PS08]

A.42.1. Abstract

The lifecycle mismatch between vehicles and their IT system poses a problem for the automotive industry. Such systems need to be open and extensible to provide customised functionalities and services. What is less clear is how to achieve this with quality and security guarantees. Recent studies in language-based security - the use of programming language technology to enforce application specific security policies - show that security policy enforcement mechanisms such as inlined reference monitors provide a potential solution for security in extensible systems. In this paper we study the implementation of security policy enforcement using aspect-oriented programming for the OSGi (Open Services Gateway initiative) framework. We identify classes of reference monitor-style policies that can be defined and enforced using AspectJ, a well-known aspect-oriented programming language. We demonstrate the use of security states to describe history-based policies. We also introduce and implement various levels of security states in Java to describe session level history versus global application level history. We illustrate the effectiveness of the implementation by deploying the security policy enforcement solution in an example scenario of software downloading in a standard vehicle system. © 2008 IEEE.

A.43. Owned policies for information security [CC04]

A.43.1. Abstract

In many systems, items of information have owners associated with them. An owner of an item of information may want the system to enforce a policy that restricts use of that information; we call such a policy an owned policy. Owned policies can be used in many contexts, including information flow, access control, and software licensing. In this paper, we introduce and study a general framework for owned policies. Relationships between security policies for a given system may be dependent on system aspects that change between or during system execution. As a result, there may be only partial knowledge of the structure of security policies available when analyzing a system statically. We demonstrate that our framework permits static reasoning about owned policies under partial knowledge, and we also exhibit tractability results for the problem of inferring security policies.

A.44. IPsec/Firewall Security Policy Analysis A Survey [KG18]

A.44.1. Abstract

As the technology reliance increases, computer networks are getting bigger and larger and so are threats and attacks. Therefore Network security becomes a major concern

during this last decade. Network Security requires a combination of hardware devices and software applications. Namely, Firewalls and IPsec gateways are two technologies that provide network security protection and repose on security policies which are maintained to ensure traffic control and network safety. Nevertheless, security policy misconfigurations and inconsistency between the policy's rules produce errors and conflicts, which are often very hard to detect and consequently cause security holes and compromise the entire system functionality. In This paper, we review the related approaches which have been proposed for security policy management along with surveying the literature for conflicts detection and resolution techniques. This work highlights the advantages and limitations of the proposed solutions for security policy verification in IPsec and Firewalls and gives an overall comparison and classification of the existing approaches. © 2018 IEEE.

A.45. Towards an automated firewall security policies validation process [AG08]

A.45.1. Abstract

A security policy constitutes one of the major actors in the protection of communication networks. However, it can be one of their weaknesses if it is inadequate according to the network security requirements. For this, a security policy has to be validated before its deployment. Unfortunately, in the literature, there is no well established validation mechanisms ensuring the well founded of such security policies. This paper proposes a validation framework for security policies based on the concept of executable specifications and applied to the firewall case. The main contributions provided by this paper concerns the adaptation of some concepts and mechanisms traditionally used in software engineering for validation aims, such as specification, executable specification or reachability graph. © 2008 IEEE.

A.46. Semantic remote attestation for security policy [ZHM10]

A.46.1. Abstract

In the software environment with security policy enforced, the behavior of application depends on not only its binary code, but also the enforcing security policy. Therefore, remote attestation for security policy is as important as that for binary code. However, since the specialties of security policy, which include more mutable and mixture of semantics, it is not suitable to use integrity measurement as the evidence to verify trustworthy of security policy. In this paper, we propose a novel semantic approach to remote attest security policy. This approach utilizes policy analysis technique to extract semantics from security policy, and utilizes logical programming to query required semantics from the mixture of semantics in security policy. We design and implement this approach on SELinux policy in this paper and discuss the security of this implementation on the basis of verified trust of TCB. ©2010 IEEE.

A.47. Policy Components - A Conceptual Model for Tailoring Information Security Policies [RKG22]

A.47.1. Abstract

Today, many business processes are propelled by critical information that needs safeguarding. Procedures on how to achieve this end are found in information security policies (ISPs) that are rarely tailored to different target groups in organizations. The purpose of this paper is therefore to propose a conceptual model of policy components for software that supports modularizing and tailoring of ISPs. We employed design science research to this end. The conceptual model was developed as a Unified Modeling Language class diagram using existing ISPs from public agencies in Sweden. The conceptual model can act as a foundation for developing software to tailor ISPs.

A.48. Security Policy Modelling in the Mobile Agent System [H19]

A.48.1. Abstract

The mobile agent security problem limits the use of mobile agent technology and hinders its extensibility and application because the constantly progressed complexity and extension at the level of systems and applications level increase the difficulty to implement a common security system as well as an anticipated security policy. Ontology is considered one of the most important solutions to the problem of heterogeneity. In this context, our work consists of constructing mobile agent domain security ontology (MASO) in order to eliminate semantic differences between security policies in this domain. We use the OWL language under the protected software to construct this ontology. Then, we chose the WS-Policy standard to model security policies, these policies are structured in forms of security requirements and capabilities. To determine the level of semantic correspondence between security policies we are developing an algorithm called "Matching-algorithm" with Java language and two APIs (Jena API and Jdom API) to manipulate the MASO ontology and security policies. © 2019 MECS.

A.49. Specification of information flow security policies in model-based systems engineering [Ger18]

A.49.1. Abstract

Model-based systems engineering provides a multi-disciplinary approach to developing cyber-physical systems. Due to their high degree of interconnection, security is a key factor for cyber-physical systems and needs to be front-loaded to the beginning of the development. However, there is a lack of model-based systems engineering approaches that enable the early specification of security policies. As a consequence, security

requirements frequently remain unspecified and therefore are hard to satisfy in the downstream development phases. In this paper, we propose to integrate model-based systems engineering with the theory of information flow security. We extend systems engineering models to information flow policies, enabling systems engineers to specify the information flow security requirements of a system under development. On refinement of the resulting models, our approach allows to derive security requirements for individual software components. We illustrate our approach using a model-based design of an autonomous car.

A.50. Automated legal compliance checking by security policy analysis [RS17]

A.50.1. Abstract

Legal compliance-by-design is the process of developing a software system that processes personal data in such a way that its ability to meet specific legal provisions is ascertained. In this paper, we describe techniques to automatically check the compliance of the security policies of a system against formal rules derived from legal provisions by re-using available tools for security policy verification. We also show the practical viability of our approach by reporting the experimental results of a prototype for checking compliance of realistic and synthetic policies against the European Data Protection Directive (EU DPD).

A.51. Security policy scheme for an efficient security architecture in software-defined networking [LK17]

A.51.1. Abstract

In order to build an efficient security architecture, previous studies have attempted to understand complex system architectures and message flows to detect various attack packets. However, the existing hardware-based single security architecture cannot efficiently handle a complex system structure. To solve this problem, we propose a software-defined networking (SDN) policy-based scheme for an efficient security architecture. The proposed scheme considers four policy functions: separating, chaining, merging, and reordering. If SDN network functions virtualization (NFV) system managers use these policy functions to deploy a security architecture, they only submit some of the requirement documents to the SDN policy-based architecture. After that, the entire security network can be easily built. This paper presents information about the design of a new policy functions model, and it discusses the performance of this model using theoretical analysis.

A.52. Closing the gap between the specification and enforcement of security policies [HPF14]

A.52.1. Abstract

Security policies are enforced through the deployment of certain security functionalities within the applications. Applications can have different levels of security and thus each security policy is enforced by different security functionalities. Thus, the secure deployment of an application is not an easy task, being more complicated due to the existing gap between the specification of a security policy and the deployment, inside the application, of the security functionalities that are required to enforce that security policy. The main goal of this paper is to close this gap. This is done by using the paradigms of Software Product Lines and Aspect-Oriented Programming in order to: (1) link the security policies with the security functionalities, (2) generate a configuration of the security functionalities that fit a security policy, and (3) weave the selected security functionalities into an application. We qualitatively evaluate our approach, and discuss its benefits using a case study. © 2014 Springer International Publishing.

A.53. PoliSeer A tool for managing complex security policies [LL11]

A.53.1. Abstract

Complex software-security policies are difficult to specify, understand, and update. The same is true for complex software in general, but while many tools and techniques exist for decomposing complex general software into simpler reusable modules (packages, classes, functions, aspects, etc.), few tools exist for decomposing complex security policies into simpler reusable modules. The tools that do exist for modularizing policies either encapsulate entire policies as atomic modules that cannot be decomposed or allow fine-grained policy modularization but require expertise to use correctly. This paper presents PoliSeer, a GUI-based tool designed to enable users who are not expert policy engineers to flexibly specify, visualize, modify, and enforce complex runtime policies on untrusted software. PoliSeer users rely on expert policy engineers to specify universally composable policy modules; PoliSeer users then build complex policies by composing those expert-written modules. This paper describes the design and implementation of PoliSeer and a case study in which we have used PoliSeer to specify and enforce a policy on PoliSeer itself. © 2011 Information Processing Society of Japan.

A.54. Modelling mobility aspects of security policies [Har+05]

A.54.1. Abstract

Security policies are rules that constrain the behaviour of a system. Different, largely unrelated sets of rules typically govern the physical and logical worlds. However, increased

hardware and software mobility forces us to consider those rules in an integrated fashion. We present SPIN models of four case studies where mobility plays a role. At present our models are ad-hoc. In each case the model captures both the system of interest and its security policy. The model is then formally checked against a security principle. The model checking activity shows examples of policies that are too weak to cope with mobility.

A.55. Security policy configuration issues in Grid computing environments [AGL03]

A.55.1. Abstract

A computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. The aim is to share among dynamic collections of individuals, institutions and resources in a flexible, secure, and coordinated manner. However, without an adequate understanding of the security implications of a Grid, both the owner who contributes resources to the Grid and the resource requestor can be subject to significant compromises in security. The basic prerequisite of establishing a secure Grid environment is the definition and implementation of a concrete security policy. Currently, in the existing Grid environments, security policy is often specified only by configuring access control lists associated with individual resources. It is obvious that significant work is needed in this area, which will lead to the specification of integrated security policies in this special environment where principals from multiple administrative domains co-exist. The purpose of this paper is to review a number of the security policies that have already been configured in existing Grid environments, identify the deficiencies and introduce a collection of all the issues that should be taken under consideration while building an integrated security policy in a Grid computing environment.

A.56. Formal security policy verification of distributed component-structured software [Her03]

A.56.1. Abstract

Component-structured software, which is coupled from independently developed software components, introduces new security problems. In particular, a component may attack components of its environment and, in consequence, spoil the application incorporating it. Therefore, to guard a system, we constrain the behavior of a component by ruling out the transmission of events between components which may cause harm. Security policies describing the behavior constraints are formally specified and, at runtime, so-called security wrappers monitor the interface traffic of components and check it for compliance with the specifications. Moreover, one can also use the specifications to prove formally that the combinations of the component security policies fulfill certain security properties

of the complete component-structured application. A well-known method to express system security properties is access control which can be modelled by means of the popular Role Based Access Control (RBAC) method. Below, we will introduce a specification framework facilitating the formal proof that component security policy specifications fulfill RBAC-based application access control policies. The specification framework is based on the specification technique cTLA. The design of state-based security policy specifications and of RBAC-models is supported by framework libraries of specification patterns which may be instantiated and composed to a specification. Moreover, the framework contains already proven theorems facilitating the formal reasoning since a deduction proof can be reduced to proof steps which correspond directly to the theorems. In particular, we introduce the specification framework and clarify its application by means of an e-commerce example.

A.57. When role models have flaws Static validation of enterprise security policies [Pis+07b]

A.57.1. Abstract

Modem multiuser software systems have adopted Role-Based Access Control (RBAC) for authorization management. This paper presents a formal model for RBAC policy validation and a static-analysis model for RBAC systems that can be used to (i) identify the roles required by users to execute an enterprise application, (ii) detect potential inconsistencies caused by principal-delegation policies, which are used to override a user's role assignment, (iii) report if the roles assigned to a user by a given policy are redundant or insufficient, and (iv) report vulnerabilities that can result from unchecked intra-component accesses. The algorithms described in this paper have been implemented as part of IBM's Enterprise Security Policy Evaluator (ESPE) tool. Experimental results show that the tool found numerous policy flaws, including ten previously unknown flaws from two production-level applications, with no false-positive reports.

B. SLR Data Extraction

B.1. Distributed Middleware Enforcement of Event Flow Security Policy [Mig+10]

B.1.1. What are the key aspects of security policies in software systems, and how are they structured?

Structure: They are separated from the functional implementations of processing units and from the enforcement mechanism at the middleware level.

Key aspects:

- The focus on maintaining the confidentiality and integrity of data.
- Encompassing high-level language policies that are designed to oversee the control of event data flows.
- Independent of specific implementation details of functional units.

B.1.2. What are the possibilities and challenges of automating the aspects of security policies in software systems?

Possibilities:

- **Event-driven Policy Enforcement:** Using an event-driven architecture to implement security policies, where the middleware itself enforces the event flow, policy actions are independent of specific processing unit implementations. This means that security guarantees can be provided throughout the system by controlling the flow of events, rather than relying on individual units to maintain security.
- **Information Flow Control:** Implement information flow control mechanisms, such as Decentralized Event Flow Control (DEFC), which involves tagging events that interact with devices. This allows the system to keep track of how data is propagated and prevent unauthorized information flow. Labels associated with events contaminate the units that interact with them, meaning that data output must include these labels, which helps with mandatory tracking of security properties for processed data.

Challenges:

- **Ensuring Correct Implementation:** There's a fundamental reliance on the assumption that the middleware itself is properly implemented and trustworthy. If this part of the system fails or is compromised, the entire security policy enforcement could be affected.

- **Potential for Error:** Traditional implementations of security policy checkers add surrounding units and may perform ad hoc policy checks at the input and output stages of events. This process can introduce human error and make it difficult to enforce security properties that depend on the behavior of a sequence of event processing units.

- **Compatibility Across Domains:** Because devices can belong to different domains that may enforce policies differently, it's necessary to manage policy enforcement in a way that respects the autonomy of each domain while maintaining the security of the overall system.

- **Complex Policy Specification:** Security policies must be both comprehensive in terms of security and efficient enough not to significantly degrade performance. This requires a balance between detailed policy specification and practical considerations of system performance.

B.2. Selection of regression system tests for security policy evolution [Hwa+12]

B.3. A Flexible Architecture for Systematic Implementation of SoC Security Policies [BBR15]

B.3.1. What are the possibilities and challenges of automating the aspects of security policies in software systems?

Possibilities:

- **Reducing Complexity:** Automating the implementation of security policies can simplify the inclusion of security measures in complex systems, such as SoCs, where policies often span multiple design blocks with intricate interactions between hardware, firmware, and software.

- **Enhancing Validation and Debugging:** Automated tools can assist in verifying that the system adheres to the defined security policies, which often poses a significant challenge given the complexity and subtle interaction patterns typical in SoC designs.

- **Facilitating Updates:** Automation can make it easier to update security policies post-deployment, which is crucial in responding to new threats that emerge over time.

Challenges:

- **Complex Policy Specification:** Security policies are typically intricate and sometimes ambiguously defined. The lack of formal, analyzable representations of these policies presents a barrier to automation.
- **Dynamic Requirements:** Security policies need to be adaptable to a system's lifecycle, including varying requirements during different stages such as manufacturing and post-production. This dynamic nature further complicates automation efforts.
- **Integration of Third-Party IPs:** Extending the architecture to include untrusted third-party IPs poses significant risks and is highlighted as an area for future work. Automated tools need to ensure that introducing such IPs does not compromise the overall security of the system.

B.4. When role models have flaws Static validation of enterprise security policies [Pis+07a]

B.4.1. What are the key aspects of security policies in software systems, and how are they structured?

Structure:

- **Sufficiency:** Ensuring that the RBAC policy is not insufficient, which can lead to stability problems due to potential run-time authorization failures.
- **Redundancy:** Verifying that the RBAC policy is not redundant, meaning it should not grant a superset of the minimal set of roles necessary to execute a program, thus adhering to the Principle of Least Privilege.
- **Subversion:** Preventing subversive executions that can bypass declared access restrictions by exploiting unchecked intra-component calls.
- **Principal Delegation:** The ability to map each component to a principal-delegation policy to override the identity of the executing principal with a specified identity, which can affect downstream calls
- **Inter- vs. Intra-component Calls:** Authorization checks are typically enforced only across component boundaries, not within a component, which can lead to security risks if access-control policies for internal execution points differ from those at the component entry point

B.4.2. What are the possibilities and challenges of automating the aspects of security policies in software systems?

Possibilities:

- Identifying the roles required by users to execute an enterprise application.
- Detecting potential inconsistencies caused by principal-delegation policies.
- Reporting if the roles assigned to a user by a given policy are redundant or insufficient.
- Reporting vulnerabilities that can result from unchecked intra-component accesses

B.5. Modeling and verification of ATM security policies with SecBPMN [SG14]

B.5.1. What are the key aspects of security policies in software systems, and how are they structured?

Key aspects:

- **Security Annotations:** Security policies in the SecBPMN Modeling Language (SecBPMN-ml) are enriched with security annotations represented by icons with a solid orange circle. These annotations depict security aspects as defined in their source literature. Each annotation is detailed with a predicate that specifies further details on the security aspects of the business process, thereby structuring the policy according to the various security needs of the system.
- **Security Policies Representation:** The SecBPMN Query language (SecBPMN-Q) is a graphical tool that helps define security policies in terms of SecBPMN-ml elements. The example provided in the paper shows how a textual security policy can be converted into a graphical representation that models the necessary conditions for security. For instance, the visa document must be authenticated and must be sent through a secure channel, ensuring protection against eavesdropping and tampering.

Structure: This graphical and structured approach allows for a clear representation of security policies and facilitates the verification of their compliance with system processes.

- Different activities in a process are labeled with symbols (e.g., @X, @Y), where the "@" symbol is used to match any activity.
- The path between activities denotes the sequence in which these activities should occur.
- Specific annotations ensure the confidentiality, integrity, and authenticity of the data being processed and transferred between system components.

B.6. Research and application of XACML-based fine-grained security policy for distributed system [SY13]

B.6.1. What are the key aspects of security policies in software systems, and how are they structured?

Structure:

- Security aspects are fundamental in modern software systems, especially those not operating in completely trusted environments. Security must be integrated early in the software development process, particularly when modeling functional requirements with UML (Unified Modeling Language)
- Access control policies based on RBAC (Role-Based Access Control) are structured with elements like subjects, roles, views, and objects. RBAC simplifies authorization management by using roles as abstractions of callers, while views describe fine-grained access rights for operations of distributed objects. Views on objects are assigned to roles, and access to an operation is permitted if there's a view granting permission; it's denied if the operation is denied or no permission is found
- Access control policies are first described by VPL (a notation used within the paper) and then presented by XACML (eXtensible Access Control Markup Language) through a mapping process. XACML provides a uniform policy description language and includes an access decision language for requests of run-time resources, allowing decisions based on attributes of the requests and policy rules. Decisions on whether a request should be allowed are classified as Permit, Deny, Indeterminate, or Not Applicable

B.6.2. What are the possibilities and challenges of automating the aspects of security policies in software systems?

Possibilities:

- Integrating access control requirements into the analysis phase of the software development process, which allows for a more seamless inclusion of security in the design
- Generating access control policies that can be described by XACML (eXtensible Access Control Markup Language) and deployed into enforcement infrastructures, facilitating standardized and automated enforcement

Challenges:

- During the software development process, security is often not sufficiently supported and tends to be considered after the design of the system, which can lead to difficulties fitting security mechanisms into a pre-existing design

- Not all access control information can be automatically generated, which means not every aspect of security can be fully automated. Sequence diagrams, for example, reflect only part of the designers' concerns and may produce the same views, thereby not covering all the access control requirements
- Despite modeling and automation efforts, human designers are still needed to utilize generated views as a basis for the ultimate access control specification, indicating a reliance on human expertise and decision-making to ensure the completeness and accuracy of security policy implementation

B.7. Enforcing Policy-Based Security Models for Embedded SoCs within the Internet of Things [Hag+18]

B.7.1. How can the aspects of security policies in software systems be tailored towards the concerns of stakeholders?

- **Flexibility:** The policy-based security model is designed to be manageable and adaptable during the device life-cycle. This means that it can be adjusted as the security landscape or stakeholder requirements change.
- **Customizable Enforcement:** Security policies can be enforced by various means, such as an operating system component like SELinux or an external hardware-based entity such as the Security Policy Engine. This allows stakeholders to choose an enforcement method that aligns with their security posture or organizational capabilities.
- **Monitoring and Response:** The policy enforcement engine features mechanisms for monitoring system-level communication and checking application permissions to identify anomalous behavior, therefore directly addressing the stakeholders' concerns about active system integrity and violation detection.
- **Independent Assurance:** By using this approach, stakeholders, such as Original Equipment Manufacturers (OEMs), do not have to rely solely on third-party vendors' security assurances because they have the means to enforce and monitor security policies derived from threat modeling of system assets and attack scenarios.
- **Cost-effectiveness:** For hardware vendors, this security approach may reduce development costs because they can offer a generic platform to multiple users. Users can then implement their own security policies based on specific security requirements, which adds a level of customizability on a per-user or per-stakeholder basis.
- **Lifecycle Management:** If security requirements change after a product's release, due to a new feature addition or the discovery of new vulnerabilities, there is a system in place for updating policy definitions to assure continued protection and adherence to changing stakeholder concerns.

B.7.2. What are the key aspects of security policies in software systems, and how are they structured?

- **Requirements Phase:** Documenting security requirements relevant to the use case, identifying system assets that may be threatened, and defining high-level mitigations to prevent the realisation of these threats.

- **Design and Development Phase:** This phase involves technical assessment of the threats, such as identifying the assets involved, confirming their security relevance, determining trust boundaries, data flows, and entry points. Threat rating methods like STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege) and DREAD (Damage, Reproducibility, Exploitability, Affected Users, Discoverability) are utilized to rate and prioritize threats, which informs the creation of countermeasures. Additionally, attack trees may be used to illustrate potential attack paths and identify components needing further protection.

- **Policy Enforcement:** The implementation phase includes the enforcement of the security policies which can be carried out either by an operating system component, such as SELinux, or an external hardware-based entity like the Security Policy Engine. This includes monitoring system-level communication to enforce security policies and checking application permissions to identify anomalous behavior.

- **Flexibility and Adaptability:** Utilizing policies allows OEM (Original Equipment Manufacturer) not to be solely dependent on third-party security assurances. Policies derived from threat modeling allow the device to operate as intended by the OEM and adapt to changes such as new features or discovered vulnerabilities after production by distributing policy definition updates.

- **Policy Enforcement Mechanism Integration:** The approach requires hardware vendors to integrate a policy enforcement mechanism to ensure the security policies are enforceable and the system operates as intended

B.7.3. What are the possibilities and challenges of automating the aspects of security policies in software systems?

- **Deriving Policies from Threat Analysis:** Automating the process of deriving and applying policies from threat analysis allows for more efficient and rapid development of security policies tailored to specific threats.

- **Lowering Costs for Hardware Vendors:** Automation can provide hardware vendors with the means to offer generic platforms that serve multiple use cases, thereby reducing the design and development costs associated with creating specialized hardware for different security concerns.

- **Mitigating Security Flaws:** Automation enables the swift application of policies to mitigate security flaws without requiring complex redesign or upgrades, which may otherwise be impractical or unacceptable for customers.

B.8. Formalizing the Relationship between Security Policies and Objectives in Software Architectures [Rou+23]

B.8.1. What are the key aspects of security policies in software systems, and how are they structured?

- The use of formal methods to formally verify and capture the satisfaction relationship between policies (security solutions) and the targeted security objectives (problems).
- The definitions of these policies and security objectives as properties, provided as formal model libraries to support their reuse.

B.9. Required Policies and Properties of the Security Engine of an SoC [MRF21]

B.9.1. What are the key aspects of security policies in software systems, and how are they structured?

- **Comprehensive Overview:** This paper provides a comprehensive overview of the requirements and components for designing a Root-of-Trust (RoT), called a Security Engine (SE), which is critical for protecting against various attacks on the manufacturing floor and during field operations. It provides the following security-critical functions and features
- **Critical Protocols and Security Policies:** This paper identifies several critical protocols and security policies for the RoT. These policies ensure secure operations and secure asset transfer while maintaining confidentiality and integrity.
- **Role of Security Policies:** Security policies are instrumental in ensuring that the principles of confidentiality, integrity, and availability (CIA) are maintained within the SoC design and its operation.
- **Critical Functions and Protocols in SoC Design:** The paper specifies critical functionalities and protocols involved in SoC design and development and describes possible solutions for each using the proposed security engine. Examples include secure boot, self-test deployment protocols, security intellectual property, watermarking, secure debug, etc.
- **Implementation and Enforcement of Security Policies:** The policies and protocols can be executed in hardware and software with minimal overhead. They

can also be audited and enforced by integrating them into the firmware code of the RoT processor.

- **Structure:** Policies and protocols are structured to be enforced against violations of security-critical protocols, potentially providing an integrated hardware and software solution that ensures systemic security and trust throughout the SoC lifecycle.

B.9.2. What are the possibilities and challenges of automating the aspects of security policies in software systems?

Possibilities:

authors have developed several security policies related to different functionalities by simulating the entities present in the protocol. This implies the possibility of using simulation and modeling techniques to develop, implement, and verify automated security policies before integrating them into real systems.Security policies, such as the secure boundary of assets, can be integrated into the firmware of a security engine (SE) to continuously monitor and verify correct behavior in the chip, ensuring that security is automatically maintained during operation.

Challenges:

the paper discusses the automation of certain security policies, it does not directly address the challenges. However, we can infer that challenges may arise from the complexity of simulating real-world entities and faithfully replicating the security environment in which these policies will operate.Automated policies must be able to deal with dynamic security threats, which means that policies must be adaptable and potentially self-modifying in response to new vulnerabilities and attacks. There is likely to be a need for high-level integration with existing systems, such as integration with SE firmware, which can be complex and may require specialized expertise. Automated policies must be thoroughly tested to ensure that they behave as expected in various scenarios, including the presence of adversarial actions.

B.10. A policy-based security model for Web system [XM03]

B.10.1. What are the key aspects of security policies in software systems, and how are they structured?

- **Definition and Functions of Security Policy:** A security policy serves several functions, including providing a forum for identifying and clarifying security goals and objectives for the entire organization. A security policy is essentially an

"aggregate set of rules" that guides each employee on how to maintain a secure environment.

- **Security Policy Structure:** The paper suggests using an XML-based schema to express the flexible rules of a security policy. An example schema provided in the paper includes elements such as <algorithm>, <authorization>, <entropy>, and <credibility> that could be part of a security policy rule.

- **Security level:** The Security Degree is defined as a measure between two sides of a communication and is represented as the minimum of several factors (e.g., Si represents the security of a single aspect of the system). These factors include the security algorithm, protocol, key management, difficulty of attacking the system, traceability, probability of detection, time required to attack, and cost of the attack.

- **Fuzzy Evaluation Model:** The paper proposes a fuzzy evaluation model to measure system security. In this model, a set of factors (U) affecting the security assessment is identified, and each factor is assigned a weight to represent its importance. An evaluation space is then constructed consisting of a triple tuple (U, V, R), where U is the set of factors, V is the set of evaluation scores, and R is a matrix of a single factor representing the fuzzy degree to which the factor contributes to the evaluation score.

B.11. FSM Modeling of Testing Security Policies for MapReduce Frameworks [HAC19]

B.11.1. What are the key aspects of security policies in software systems, and how are they structured?

- **Authentication, Authorization, and Access Control:** It's necessary to properly authenticate and control access to mappers and reducers to prevent unauthorized use.

- **Availability:** Ensuring that data, mappers, and reducers are available to authenticated and authorized users without delay is critical for consistent operations.

- **Confidentiality:** This includes protecting computations and data, whether in transit or stored in public clouds, from unauthorized users and the public cloud providers themselves.

- **Integrity:** This refers to the accurate and fair transfer and execution of MapReduce computations, preserving the original state of the data without unauthorized changes.

B.12. Trust-adapted enforcement of security policies in distributed component-structured applications [HK01]

B.12.1. How can the aspects of security policies in software systems be tailored towards the concerns of stakeholders?

- **Explicit Contracts:** Each component integration into the software system is accompanied by a contract describing the agreed properties of the component, in particular its interface operations and relations with its environment. This contract shall include the security-related behavior expected of the component during its execution.
- **Design-Time Analysis:** At the time of system design, the structure of the system is analyzed in combination with the behavioral descriptions of its components. This is done to prove that the required security properties of the system will be maintained if each component behaves according to its contract.
- **Runtime Behavior Control:** At runtime, it is critical to confirm that the actual behavior of each component is consistent with its contract, to prevent security breaches caused by malicious components or compromised code. This can be accomplished by using wrappers that monitor and control the behavior of each component.
- **Trust-Aware Control Functions:** Detailed control functions can introduce significant overhead. Therefore, the intensity of monitoring and behavioral checks can be adjusted based on the level of trust the component has in the eyes of the application owner.
- **Stakeholder-Specific Security Goals:** The security model used should identify a rich set of principals that reflect stakeholder objectives, including the confidentiality, integrity, accountability, and availability properties of data and functions required by different principals.
- **Dynamic Adaptation:** The intensity of control functions, such as monitoring and behavioral checks, is dynamically adapted to the current level of trust the system has in the component, its hosting environment, and its vendor. This addresses the concerns of different stakeholders who may have different levels of trust and security priorities.

B.12.2. What are the key aspects of security policies in software systems, and how are they structured?

- Local applications, distributed applications, mobile code applications, and distributed component-based applications.
- Local applications focus on defining user classes, authentication, and access control, assuming trusted software and trusted hosting systems.

- Distributed applications recognize the untrusted nature of networks connecting disparate host computers, requiring secure communication services, distributed authentication, and access control systems.
- Mobile code applications introduce the need for secure transfer of code modules and protection against malicious code as code migrates between hosts.
- Distributed component-based applications consist of components provided by different vendors, introducing new principles and the need for protection against malicious components, hosting environment failures, unauthorized component use, and accountability for actions.
- Principals include users, resource owners, application owners, host providers, and component vendors.
- Objects include resources, software components, application configurations, hosting environments, and communications facilities.
- Relationships include the components that make up applications and contribute to functionality, with components accessing and managing resources on behalf of a user and passing information and control to other components.

B.12.3. What are the possibilities and challenges of automating the aspects of security policies in software systems?

Possibilities:

- **Security Automata:** Security policies can be formally modeled using security automata, where an automaton can enforce a policy by simulating it simultaneously with the execution of code. This allows the code to perform an execution step only if it corresponds to a transition of the automaton.
- **Code Instrumentation:** Methods such as software fault isolation ensure that untrusted code is executed and monitored in a safe part of the system, preventing damage from attacks. This is a form of automated security that monitors the behavior of code as it runs.
- **Language-Based Security:** Security-related information about mobile code is obtained, allowing the program user to check the code for compliance with their security policies.
- **Verifiable Code:** Enables formal program verification, where the program developer annotates the code with a formal specification and the user proves the code's compliance.
- **Trust Management:** The mathematical expression of trust in a human or computer principal, taking into account past experience and enabling authorization systems to grant access based on credentials issued by trusted third parties.

- **Trust Adaptive Enforcement:** Incorporating generic wrappers that integrate security automata for state-dependent security constraints and manage the analysis without the need for explicit formal verification.

- **State-Dependent Security Constraints:** Can be modeled by state automata and checked for compliance at run time, recording the history of events and ensuring that pending events are compliant.

Challenges:

- **Code Trustworthiness:** The problem arises when component code cannot be fully trusted. Methods such as proof-bearing code and code verification rely on information provided by the code developer, which could be distorted to hide malicious code.

- **Runtime Verification:** Generic software wrappers check code for security properties at runtime without modifying it, but ensuring that these checks are accurate is a challenge.

- **Dynamic Behavior Control:** Supporting dynamic behavior control can be complex because it's not always feasible to enable or disable automata, especially if events during disabled periods are relevant to later enabled periods.

- **Complex State Checks:** The execution of some aspects of security policies, such as the computation of more complex state checks, may depend on the activation state of the automaton and require continuous tracking.

- **Compatibility with existing policies:** Before deploying a component, users must check the specifications for compliance with the security policies of the system to which the component will belong.

- **Specification and Conformance:** Security specifications, which are usually developed by a component vendor, must be compliant with the security policies of the system. This requires close scrutiny by the component user.

B.13. An Automated Validation Method for Security Policies the firewall case [AE08b]

B.13.1. What are the key aspects of security policies in software systems, and how are they structured?

- **Definition and Structure of Security Policies (SPs):** Security policies are formally defined as "rules that people who are given access to an organization's technology and information assets must follow".

- **Executable SPs Specifications:** This paper discusses Executable Security Policies (ESPs), which are models of SPs that can generate the expected behavior of a secured system. These models use processes and channels to represent the behavior of active entities and data transfers between processes. The system state is defined by the state of the output channels of the SP process at a given time, and this information follows a First In, First Out (FIFO) discipline for data transfer through the channels.

- **Validation Process for SPs:** The security policy validation process consists of three steps and focuses on verifying that The SP model is mathematically consistent, i.e. there are no contradictions within the specified rules. The security enforcing functions maintain the desired security properties. The SP model is complete with respect to its input space.

B.14. Policy-based security channels for protecting network communication in mobile cloud computing [IKC11]

B.14.1. What are the key aspects of security policies in software systems, and how are they structured?

- **General Identification Section:** This section provides identification information such as the type of policy (default policy provided by the SVM or a custom policy updated by the customer) and identifies the Service ID, CSP ID, and Customer ID.

- **Security Algorithms Section:** Lists the specific security algorithms to be used to ensure data confidentiality and integrity. It may include algorithms for encryption, Message Authentication Codes (MAC), and data encryption algorithms.

- **Protection Rules Section:** This section outlines the scope and level of security operations to be applied to cloud data before it leaves the SVM on the CSP (Cloud Service Provider) side. The scope determines which fields in the records should be encrypted and MACed based on the sensitivity of the field contents (such as medical records, social security numbers, etc.). The level indicates the strength of the encryption to be applied. Higher key lengths correspond to increased security as well as increased processing requirements. The policy enforcement engine secures each field based on the protection rule it matches. The matching criterion is the field content that matches a specific regular expression. Each protected field is then tagged with the rule ID to ensure proper decryption and integrity verification by the client's policy enforcement engine according to the policy file.

B.15. Verify consistency between security policy and firewall policy with answer set programming [LD08]

B.15.1. How can the aspects of security policies in software systems be tailored towards the concerns of stakeholders?

- **Demonstration to Management:** A firewall administrator can use this method to demonstrate to his or her manager that the firewall is in compliance with the organization's security policy. This is invaluable for accountability and ensuring compliance with established security policies.

- **Using Answer Set Programming (ASP):** The method is rooted in Answer Set Programming, a declarative programming paradigm that can represent knowledge. This means that the method can be understood by non-programmers, such as security administrators.

- **Applicability Across Firewall Vendors:** Because ASP is a general formalism, the method is not vendor-specific and can be applied to different firewalls from different vendors. This flexibility ensures that the method can accommodate the different security infrastructures that exist in different organizations.

- **Comprehensibility and practicality:** Because of the logical language used in the method, it is comprehensible to security administrators, enabling them to understand and possibly modify security policies and their implementation. The method provides a unique answer that can be computed in polynomial time, indicating that it is feasible to use in a real-world work environment without requiring excessive computational resources.

B.15.2. What are the key aspects of security policies in software systems, and how are they structured?

- **Security Policy as High-Level Design:** This is a high-level policy that defines the types of access that are allowed or denied from the network. Viewed as a design of network security, it defines what services are allowed or denied.

- **Firewall policy as implementation code:** A lower-level policy that details how the firewall will implement the security policy. Viewed as a set of codes that execute the high-level design provided by the security policy.

- **Consistency Verification:** The effectiveness of firewall security depends on the firewall policy being consistent with the security policy. This paper presents a method for verifying the consistency between these two policies using model checking. Two policies and their consistency are represented by logic programs. Verification is performed by testing whether the logical formula of consistency is satisfied within the semantics of these logic programs. It is essential to identify any

inconsistencies between the security policy and the firewall policy to ensure that the intended security design is correctly implemented at the firewall level.

- **Computational Aspect:** The method presented in this paper guarantees a unique answer and can be computed in polynomial time, suggesting that it is efficient in terms of computational resources.

B.15.3. What are the possibilities and challenges of automating the aspects of security policies in software systems?

- **Policy Design and Errors:** Many firewall policies are poorly designed and have numerous bugs, making automation difficult.

- **Complex Firewall Policy Languages:** Firewall policy languages are often obscure, highly vendor-specific, and very low-level. This makes them difficult to deploy and automate across platforms.

- **Policy Order Sensitivity:** Firewall policies are sensitive to rule order; often the first matching rule is applied to incoming traffic. Therefore, changing the order of rules or misplacing a rule can lead to unexpected behavior and potentially create security vulnerabilities.

- **Large number of rules:** Firewalls typically have a large number of rules, which increases complexity and makes it cumbersome for administrators to manually check for consistency between security and firewall policies.

B.16. Analysis of Policy-Based Security Management System in Software-Defined Networks [Soo+19]

B.16.1. What are the key aspects of security policies in software systems, and how are they structured?

- Intelligent traffic control for both generalized traffic management and flow-based security management.

- Maintenance of QoS using deep packet inspection, queuing mechanisms, and load balancers.

- Ability to identify fine-grained, secure traffic flows to detect end-user metadata at granular levels.

B.17. Managing security policy in a large distributed Web services environment [CCH03]

B.17.1. How can the aspects of security policies in software systems be tailored towards the concerns of stakeholders?

- **Dynamic Establishment of Security Policies:** Security policies must be established at the start of each application instance, addressing the concerns of stakeholders who require application instance-specific and timely security measures.
- **Flexibility for Location of Security Operations:** The security policy should be flexible enough to specify where critical security operations, such as encryption/decryption and signing/verification of signatures, should be performed. This can address stakeholder concerns about key management efficiency and the desire for either centralized or end-to-end security.
- **Online Dynamic Negotiation:** For session-based Web services, SSL's dynamic negotiation approach can adapt to the security preferences of the parties involved, which can satisfy stakeholders who require adaptable and secure communication channels.
- **Centralized Configuration Approach:** Although this approach has vulnerability concerns, it suggests that stakeholders are interested in having security policies that are easily managed and centrally located, but with secure measures against unauthorized access.
- **Trusted Third Party Approach:** This approach introduces a trusted intermediary server that handles security processing according to pre-registered security policies, addressing stakeholders who prioritize end-to-end agreement on security policies, but also highlighting concerns about scalability and bottlenecks.
- **ICD (Interoperability Contract Document) Approach:** The solution proposed in the document bases the security agreement on an ICD that is attached to messages during interoperation. Each business partner registers its service security profile in a registry and before each transaction or message; the sender generates an ICD after comparing the profiles, which then defines the security policy for the communication. This approach addresses stakeholders who require dynamic, version-controlled, and flexible policy management capable of many-to-many collaboration without ambiguity.

B.17.2. What are the key aspects of security policies in software systems, and how are they structured?

- **End-to-End Agreement for Many-to-Many Interoperation:** Security policies must support interoperability across multiple hops, steps, and users while ensuring end-to-end agreement on security policies for each transaction.

- **Dynamic Factors Impacting Security Policies:** Changes in security policies and algorithms, changes in document schema, updates to integrity and confidentiality requirements, and expiration of public certificates must be accommodated.
- **Version Interoperability:** Long-term transactions should not be disrupted by security policy changes, requiring versioning support to handle updates seamlessly.
- **Interoperability Privacy:** The security policy for each business transaction or individual communication should remain confidential.
- **Dynamic Establishment of Security Policies:** Security policies should be established at the start of each application instance, as statically defined security policies become inappropriate over time.
- **Automated Negotiation:** Sending and receiving parties must negotiate security algorithms and key sizes, possibly automating this process to ensure that the set of supported algorithms intersects successfully.
- **Flexibility in the Location of Security Operations:** The security policy should allow flexibility in determining where encryption, decryption, signing, and signature verification take place, which can enable end-to-end security or a more centralized approach.

B.18. A model for the analysis of security policies in service function chains [Dur+17]

B.18.1. How can the aspects of security policies in software systems be tailored towards the concerns of stakeholders?

- A Traffic Monitor (TM) implements the policy to count connections to the database.
- An application firewall (AF) ensures that all encrypted outbound traffic is dropped to prevent the undetected disclosure of confidential information.
- A VPN Gateway (VG) enforces the policy to encrypt all traffic to the data center.

B.18.2. What are the key aspects of security policies in software systems, and how are they structured?

- **Policy Enforcement Verification:** Verifying the correct enforcement of security policies within SFCs is critical due to new challenges such as properly configuring and ordering service functions (SFs) to meet overall security requirements.
- **Handling Misconfigurations:** Traditional configurations of service functions often rely on low-level parameters that are set manually, resulting in trial-and-error approaches. Misconfigurations can result in ad hoc rules that increase complexity

and maintenance difficulty, highlighting the need for more systematic approaches to configuration.

- **Interactions Between Functions:** Ensuring security across the network requires careful consideration of the interactions between different network security functions (NSFs), especially in distributed environments.
- **Automated Software Tools:** Given the size and complexity of networks, preventing conflicts in SFC configurations is challenging without automated software tools. Practical solutions require formal descriptions and sound theoretical foundations.
- **Formal Model for Traffic Transformations:** The paper proposes a formal model capable of describing typical actions performed to process network traffic, such as packet header modification and payload encryption, considering both individual SFs and their sequential combinations.

B.19. TechNETium Atomic Predicates and Model Driven Development to Verify Security Network Policies [Ber+20]

B.19.1. What are the key aspects of security policies in software systems, and how are they structured?

- **Policy Definition:** In the context of this paper, a "policy" refers to a high-level objective, expressed as a rule or configuration, that is intended to achieve certain desirable states or behaviors in the network. Policies are defined and described by users or network administrators.
- **Formal Verification of Policies:** TechNETium provides formal verification of network forwarding rules by interacting with the SDN network controller. This process involves verifying that the defined set of policies is satisfied by the network model.
- **Atomic Predicates and Binary Decision Diagrams (BDDs):** The tool uses atomic predicates to efficiently verify the fundamental property of network reachability. BDDs are used to represent reachability in the network model.
- **Network Model and Property Verification:** TechNETium maintains a model of the network and continuously verifies that the policy set is satisfied. Security policies are seen as properties that can be checked against this network model.
- **Efficient Policy Verification:** Security policies, such as reachability and waypoint routing, are decomposed into elementary graph operations that are then verified on the BDD representation of the network.

- **Scalable and Portable Implementation:** TechNETium is designed to be portable and architecture independent, written in Java for ONOS, a widely used SDN controller.
- **Use Cases:** TechNETium can be used proactively or reactively by network administrators to define security policies and manage network security, for example to counter common network attacks such as Distributed Denial of Service (DDoS).

B.20. Analyzing RBAC Security Policy of Implementation Using AST [PTN09]

B.20.1. What are the key aspects of security policies in software systems, and how are they structured?

- **Security Policy Criticality:** Security policy is a critical aspect of software applications that require a high level of security.
- **Requirements and Implementation:** Security policies must be clearly specified in requirements documents, and the implementation must conform to the specifications.
- **Access Control:** Access control is a security mechanism that grants official permissions to entities to perform specific activities, with the goal of maintaining the confidentiality and integrity of the system.
- **RBAC Model and Users:** In RBAC, users are people who interact with a computer system. A user can invoke multiple computer processes, called subjects in RBAC, and all activities must be checked against the user's privileges or roles.

B.20.2. What are the possibilities and challenges of automating the aspects of security policies in software systems?

- **Complex Systems:** The paper recognizes that designing and implementing security policies is always a difficult task, especially in complex systems.
- **Compliance Checking:** The authors propose an approach to checking compliance between a system's security policy specifications and its implementation.
- **Using RBAC:** The use of Role-Based Access Control (RBAC) as a new and efficient language based on user roles is proposed to specify user permissions of security policies. This illustrates a way to automate the specification of access control in a structured way.
- **Source code analysis:** Another possibility for automation is to analyze the Abstract Syntax Tree (AST) of source code to check for consistency. The authors suggest that their method is suitable for real systems that only provide source code for static analyzers to examine.

- **Conformance Testing:** The paper cites work by Ammar et al. that addresses the problem of conformance testing for Access Control Implementation under Test (ACUT) in Temporal Roles-Based Access Control (TRBAC) systems, providing

B.21. A flexible architecture for security policy enforcement [AE08a]

B.21.1. How can the aspects of security policies in software systems be tailored towards the concerns of stakeholders?

- Subject: Active entities within the system, such as human users, employees, processes, applications, or programs.
- Object: Passive entities within the system, also called assets or resources, such as ports, data, or hosts.
- Operation: Actions that a subject can perform on an object, including connections, read requests, and write requests.
- Security Rules: Policies that express the appropriate security decisions (allow or deny) for each operation attempt, distinguishing between legal and illegal actions.
- Authorization rule: Distinguishes authorized from unauthorized operations by subjects. It is considered a request that expects a response.
- Obligation Rule: Actions that a subject is obligated to perform according to specific security constraints.
- Negative Obligation Rule: Actions that a subject must refrain from performing because they are prohibited by security constraints.

B.22. Using security policies in a network securing process [AF11]

B.22.1. What are the key aspects of security policies in software systems, and how are they structured?

- **Security Requirements Definition:** This phase is used to identify any necessary requirements for handling expected interactions between subjects and objects within the network.
- **Security Policy Specification:** Once the security requirements have been defined, they must be formalized using an appropriate formal specification language, resulting in a formal SP specification. At this stage, due to human intervention, the SP specification may contain errors, omissions, and inconsistencies, requiring validation activities.

- **Validation Activity:** The validation activity ensures that the SP specification is the correct one for the desired security requirements. It involves checking that the SP meets the security requirements through a process that should be repeated as necessary until the specification is declared valid.
- **Test Activity:** A validated specification is then prepared for deployment to the network. Before actual deployment, it's critical to verify that the specified SP meets all the necessary and desired security requirements, typically through testing. This paper proposes to generate elementary tests for a given SP according to the desired functionality.
- **Security Policy Deployment:** Once validated and tested, the SP is ready to be deployed on the network.

B.23. Towards policy unification for enterprise network security [YZG17]

B.23.1. What are the key aspects of security policies in software systems, and how are they structured?

- **Various Specialized Devices:** Enterprise network security has traditionally involved a variety of specialized devices (called "middleboxes"), each of which provides specific functions such as network access control, vulnerability assessment, intrusion detection and prevention systems, and firewalls. These devices share the common goal of securing the network, but vary widely in functionality, policy constructs, configuration, and enforcement.
- **Policy Unification Challenge:** Network operators manage numerous lines of configuration across these devices and must coordinate the disparate functions for effective network security. This paper addresses the challenge of unifying policy enforcement across these disparate functions.
- **Software-Defined Networking (SDN):** The researchers presented an architecture based on SDN that aims to unify policy enforcement. SDN enables remote programming of the data plane through logically centralized control functions, making the network more programmable.
- **IEEE 802.1X Extension:** The architecture extends the IEEE 802.1X framework and abstracts the connectivity context of network endpoints, enabling cross-functional policy composition and enforcement. This allows for simple yet effective policy definitions that span different specialized network security functions.
- **Role-Based Policies:** Policy in the SDN-based architecture is defined using high-level principal names in the form of role-based rules. Policy enforcement is dynamic and immediate, managed by the SDN controller.

- **Authorization State Extension & Policy Composition:** The paper suggests extending the authorization state with an authzState variable that includes states such as Unknown, Healthy, Quarantined, and Blacklisted. This allows for more comprehensive tracking and unification of network security policy enforcement, including both network access control and network admission control.

B.24. Policy Based Security Middleware as a Service [Chr+14]

B.24.1. How can the aspects of security policies in software systems be tailored towards the concerns of stakeholders?

- **Identification and Activation:** Each policy is identified by an ID attribute and can be activated or deactivated by client application owners using the active tag, allowing stakeholders to control which policies are in effect and when they apply.
- **Enforcement and Accountability:** The subject in the policy, the Policy Enforcement Point (PEP), is responsible for enforcing the action on the target entity. To enforce accountability, there is an audit tag to keep track of which policies have been triggered.
- **User-Friendly Conditions:** The conditions that trigger policies are designed to be user-friendly and are linked using first-order logic. This makes it easier for stakeholders to specify conditions without deep technical knowledge.
- **Context Awareness:** Conditions can incorporate contextual information to enforce security, such as time of day, user roles, or specific environmental factors. This allows stakeholders to express security policies that are dynamic and responsive to the context in which the system is operating.
- **Flexibility and customization:** Policies can be customized to meet different security needs, such as privacy, integrity, or confidentiality. The flexibility of the system ensures that stakeholders can adapt their policies as their security needs evolve.

B.25. Enforcing multilevel security policies in database-defined networks using row-level security [AA19]

B.25.1. What are the key aspects of security policies in software systems, and how are they structured?

- Automate orchestration of abstractions for network control.
- Facilitate ad hoc programmable abstractions through database views.
- Enable new applications to be constructed and built upon.

- Express integrity constraints and high-level policy constraints through SQL statements.
- Provide an SQL interface for viewing and updating network health and configuration, allowing the network to be queried and controlled via SQL commands.

B.25.2. What are the possibilities and challenges of automating the aspects of security policies in software systems?

- **Defining policies in plain language:** Policies can be described in a plain-language format. Administrators can define data flows in terms of matching fields and identify which security properties should apply, creating an accessible way to set and customize policies.
- **Automatic Conversion and Enforcement:** Once policies are defined, they are automatically translated into a set of rules that are implemented at the network device level. This automation streamlines the security enforcement process and ensures consistency across the network.
- **Predictive security:** Administrators can specify security levels that dictate how the system should respond to malicious traffic, enabling proactive responses to security events.

B.26. Evaluating the Use of Security Tags in Security Policy Enforcement Mechanisms [Alv+15]

B.26.1. What are the key aspects of security policies in software systems, and how are they structured?

- **Always invoked:** Permissions are checked upon every operation to ensure that decisions are based on current permissions and security labels, not outdated information.
- **Non-bypassable:** It's not possible to circumvent the access control mechanisms and gain direct access to resources.
- **Tamperproof:** The security system and its metadata must be protected against unauthorized changes.
- **Evaluatable:** The security mechanisms must function as specified and should be separable from non-security related code areas to provide high confidence in the system's security.
- **Type enforcement:** This approach uses tags to distinguish code from data and system data from user data; it also places type information on the data to restrict the operations that hardware can perform on it, thereby preventing attacks such as buffer overflows.

- **Runtime Semantic Violations:** These violations occur when user data is evaluated in an unexpected context. Security tags can prevent such attacks by distinguishing user data from command data, such as in SQL injection attacks.

B.27. A novel approach for integrating security policy enforcement with dynamic network virtualization [Bas+15]

B.27.1. How can the aspects of security policies in software systems be tailored towards the concerns of stakeholders?

- **User-Oriented Security Specifications:** The Policy Manager provides a user interface that helps stakeholders define security policies without requiring detailed technical knowledge. This interface can accept policies expressed in high-level terms, such as "do not download malware" or "do not access blacklisted sites," that are easy for non-technical stakeholders to understand and formulate.

- **High-level policies (HLP):** HLP is an authorization language designed to capture stakeholder security requirements in a technology and implementation agnostic manner. This allows stakeholders to focus on their security concerns and desired outcomes, rather than the underlying technology used to enforce those policies.

- **Policy Abstraction Layers:** By creating abstraction layers (HLP, MLP, and concrete VNF configurations), the system allows stakeholders to engage at the level most appropriate to their expertise and concerns. For example, a business owner could be involved at the HLP layer to set broad security goals, while IT professionals could delve into the details of the MLP or concrete VNF configurations.

- **Policy Refinement Process:** The translation of high-level policies into concrete configurations, performed by the Policy Manager, ensures that stakeholder concerns are systematically addressed and appropriately implemented. The policy refinement process bridges the gap between policy intent and technical enforcement.

- **Automation and Optimization:** The Policy Manager automates the selection and configuration of appropriate VNFs, including optimization to select the most appropriate security features when multiple options are available. This ensures that security measures are not only tailored to stakeholder needs, but also implemented efficiently.

- **Non-Enforceability Analysis:** To avoid mistakes and ensure that stakeholder requirements are feasible, the Policy Manager performs a non-enforceability analysis. This analysis verifies that the selected VNFs can meet the user's security requirements and informs the stakeholders and provides remediation tips if there are any problems.

B.27.2. What are the key aspects of security policies in software systems, and how are they structured?

- **Policy Manager:** A new software component added to the NFV architecture that allows users to specify their security requirements in an easy-to-use manner. It automates the process of provisioning, deploying, and configuring security features, hiding the complexity from the user.

- **Policy Refinement Techniques:** The Policy Manager uses these techniques to translate user-defined high-level policies into operational policies that can be enforced by the system. This process allows the policy specification to be separated from the implementation details of various security features.

- **High-Level Policies (HLP):** An authorization language used to define user security requirements in a way that resembles natural language sentences. These policies are technology, feature, and implementation agnostic, making them suitable for capturing user intent without being tied to specific technologies or features.

- **Translation into concrete configurations:** The Policy Manager refines HLP policies into concrete configurations for each Virtual Network Function (VNF). It identifies which VNFs are required to enforce the user's security requirements and derives their configurations, selecting from multiple available VNFs to best satisfy the same requirement.

- **Policy Abstraction Layers:** The approach divides the refinement process into two steps, using an intermediate format called Medium-Level Policies (MLPs). There are three policy abstraction layers - HLP, MLP, and the concrete VNF configurations. The first translation module refines HLP into MLP, and the second translates MLP into concrete VNF configurations.

- **Conformance and Error Avoidance:** To ensure compatibility between the user's requirements and the selected VNFs, the Policy Manager performs a non-enforceability analysis, providing the user with indications and remediation tips in case of discrepancies.

B.27.3. What are the possibilities and challenges of automating the aspects of security policies in software systems?

Possibilities:

- **Automating the Provisioning of Security Functions:** By integrating a policy manager into the NFV architecture, the paper demonstrates that it's possible to automate the deployment and configuration of security functions such as firewalls, traffic inspection systems, or VPN concentrators. This reduces the time and cost of deploying security measures.

- **Dynamic Configuration:** Leverages Virtual Network Functions (VNFs) to dynamically add and remove network security functions to adapt to changing security requirements and threat landscapes in real time.
- **Policy Abstraction:** Policy abstraction layers, such as High-Level Policies (HLP), enable non-technical stakeholders to express security requirements in natural language, facilitating broader participation in security policy specification.
- **Easy-to-use security management:** With the Policy Manager, stakeholders can easily specify their security requirements and the system takes care of the detailed configuration, providing a user-friendly approach to security policy management.
- **Consistency and Compliance:** Automated tools, such as Policy Manager, can ensure that deployed security configurations consistently reflect stakeholder specifications, helping to ensure compliance with security policies.

Challenges:

- **Complex Management Tasks:** The adoption of NFV-based approaches, while beneficial for managing security functions, increases overall management complexity. This complexity needs to be managed effectively, possibly using sophisticated policy management systems.
- **Policy Refinement:** Translating high-level policies into operational policies that can be enforced by the system, while ensuring that they meet the requirements of the high-level policy, presents a challenge in ensuring the accuracy and robustness of the refinement process.
- **Optimization Among Multiple VNFs:** When multiple VNF options are available to satisfy the same requirement, deciding which VNFs to use requires optimization. The system must balance factors such as performance, cost, and compatibility.
- **Non-Enforceability Analysis:** Ensuring compatibility between user requirements and the selected VNFs, and providing stakeholders with indications and remediation tips when there are discrepancies, requires sophisticated analysis capabilities.
- **Integration with Existing Systems:** Integrating an NFV-based policy manager with existing network and security infrastructure is likely to present significant integration challenges, including compatibility with legacy systems and existing workflows.
- **Security of the Policy Manager Itself:** As the central component responsible for security policies, ensuring the security and robustness of the Policy Manager is critical to prevent it from becoming a single point of failure or an attractive target for attackers.

B.28. Compile-time enforcement of dynamic security policies [Eye+08]

B.28.1. What are the key aspects of security policies in software systems, and how are they structured?

- **Dynamic Security Policies:** The software system must explicitly manage the security system state at runtime. This is necessary to implement dynamic constraints such as delegation, dynamic separation of duties, and other dynamic security constraints.

- **Role Based Access Control (RBAC):** RBAC is an effective way to express policy in capability systems because it introduces indirection between principals (e.g., users) and the privileges they want to access, which are grouped into roles that can be activated within sessions. RBAC therefore allows the abstraction of the identities of individuals from the collections of privileges they require, facilitating the evolution of security systems as roles change within an organization.

- **Policy Specification:** Security configuration is specific to each organization, and the paper suggests that representing the deployment-specific configuration in a policy specification separate from the application software itself allows the policy specification to evolve independently.

- **Static vs. Dynamic Constraints:** Static constraints, such as static segregation of duties, are not affected by session state, whereas dynamic constraints, such as dynamic segregation of duties, vary with session state and context.

- **Using the Kilim Framework:** The paper presents research using the Kilim framework, which provides static compile-time type linearity checking and enforces the requirement that clients do not retain access to the credentials they submit to policy decision points (PDPs).

- **Capability Systems:** In the capability systems discussed, clients provide exclusive control of their submitted credentials to PDPs, and based on the policy decision, PDPs augment the client's credentials with additional capabilities.

- **Future Work:** The paper outlines future work that discusses distributed operations, transient delegation, dynamic separation of duties, and handling of system failures. The intention is to use the Kilim framework in both single and distributed system environments to enforce security constraints and handle the complexities of distributed capability architecture, including the use of leased certificates.

B.29. OpenSec Policy-Based Security Using Software-Defined Networking [LR16]

B.29.1. How can the aspects of security policies in software systems be tailored towards the concerns of stakeholders?

- **Human Readable Policy Specification Language:** OpenSec aims to provide a language that allows security policies to be easily specified without requiring technical expertise in OpenFlow or low-level network configurations. This approach allows stakeholders with different technical backgrounds, such as network operators or security officers, to effectively define security policies.
- **Automated Response to Security Alerts:** OpenSec's ability to automatically respond to security alerts can be seen as a mechanism to address the operational concerns of stakeholders. By automating responses to security threats, stakeholders can be assured that the system will respond quickly and efficiently to potential attacks without the need for continuous manual monitoring.
- **Modularity and Dynamism in Policy Management:** OpenSec details how policies can dynamically update forwarding rules in response to security alerts. This flexibility allows stakeholders to quickly adapt and update their security strategies as their concerns evolve.

B.29.2. What are the key aspects of security policies in software systems, and how are they structured?

- **Creating and Implementing Security Policies:** OpenSec is an OpenFlow-based security framework that enables the creation and implementation of security policies using a human-readable language. This approach simplifies the task for network security operators, who can now focus more on policy specification rather than device-level configuration.
- **Security Services and Threat Response:** With OpenSec, users can describe network flows in terms of OpenFlow matching fields and define which security services should be applied to each flow (e.g., deep packet inspection, intrusion detection, spam detection, etc.). In addition, users can specify security levels that determine how to respond when malicious traffic is detected, such as alerting, quarantining traffic, or blocking packets from a specific source.
- **Scalability and Automation:** The framework is scalable and can detect up to 95% of attacks and automatically block 99% of malicious source nodes in an existing dataset. It reduces the need for manual intervention by automatically responding to security alerts specified by policy.
- **Human Readable Policy Language:** The policy specification language in OpenSec is designed to be simpler and allow for faster translation times compared

to existing solutions, making it more efficient and user-friendly for specifying security policies.

B.29.3. What are the possibilities and challenges of automating the aspects of security policies in software systems?

Possibilities:

- **Human-Friendly Configuration:** OpenSec aims to be a dynamic and automated security framework that prioritizes human-friendliness. This means that security operators can work at a higher level without having to worry about the complexity of low-level network configurations.
- **Automated Response to Security Events:** OpenSec is designed to automatically respond to security events. When suspicious traffic is detected by security services (such as firewalls, intrusion detection systems, etc.), OpenSec can modify traffic rules according to the pre-defined policy, potentially minimizing the need for human intervention.
- **Simple Policy Specification Language:** OpenSec's policy specification language is designed to be simple and accessible, allowing operators to direct traffic to the necessary security processing units and enable automated responses to security events.
- **Customization of Security Policies:** Operators can use human-readable policies to customize the security of the network and dictate how the controller should automatically respond when malicious traffic is detected, tailoring security measures to the specific needs of the network.

Challenges:

- **Maintaining Performance and Reliability:** One challenge in automating security policies is ensuring that network performance and reliability are not compromised. OpenSec addresses this by moving middleboxes away from the main data path, ensuring that traffic that needs to be processed by security services does not interfere with regular traffic flows.
- **Scaling Security Services:** Scalability is a critical challenge when designing automated security frameworks. OpenSec aims to address this challenge by offloading traffic analysis to dedicated processing units rather than the centralized controller to mitigate potential bottlenecks.
- **Integration with existing network events:** The framework must also address how to integrate and respond to network events. While OpenSec automates responses to security alerts, other frameworks such as Procera and Fresco allow for granular control of flow setup, which could challenge OpenSec in handling more complex network conditions.

- **Complexity of Policy Enforcement:** OpenSec must also maintain simplicity in security policy enforcement. This simplicity can potentially conflict with the need for nuanced responses to different types of security incidents, and may require a balance between automation and granular control.

B.30. A software architecture for automatic security policy enforcement in distributed systems [HBM07]

B.30.1. How can the aspects of security policies in software systems be tailored towards the concerns of stakeholders?

- **Policy Specification Language:** The language should be designed with careful consideration of the security domain and the various tools, specifications, and frameworks used by stakeholders. The language proposed in this document, PPL (Policy Programming Language), is a domain-specific language that encapsulates the core elements necessary for security policy definition and enforcement.

- **Modularity and Composition:** The language must support modular construction to handle complex systems and allow decomposition into manageable components, ensuring that policies can be tailored to specific areas of stakeholder interest and composed into a comprehensive policy.

- **Interface language for reuse:** By using an interface language, existing security action libraries can be effectively reused, meeting the needs of stakeholders who require integration with legacy systems or predefined security mechanisms.

- **PPL Core Concepts:** The language is based on core concepts such as Entities, Scopes, Rules, Actions, and Policies. These building blocks allow stakeholders to define how subjects (such as users or processes) may access objects (such as files or devices) under specified circumstances.

- **Scope Abstraction:** Scopes are critical for achieving compactness, generalization, and scalability in policy specification. By grouping entities together, rules don't need to be repeated for each entity individually, addressing stakeholder concerns about efficiency and simplicity.

- **Policy Types:** The language supports multiple policy types to meet the diverse needs of stakeholders, such as authorization, obligation, and delegation policies, as well as compound policies for large, distributed systems.

- **Existing Frameworks and Tools:** Analysis of existing frameworks, such as Ponder and PDL, that provide specific capabilities for security policy specification, integration, and enforcement helps to align policy specification with stakeholder concerns and operational practices.

B.30.2. What are the key aspects of security policies in software systems, and how are they structured?

- **Ponder:** This is a declarative, object-oriented language used to specify management and security policies, allowing the expression of authorizations, obligations, information filtering, withholding policies, and delegation policies. It provides a way to describe rules that can constrain the behavior of components.

- **ASL (Authorization Specification Language):** A formal logical language for specifying access control policies, including meta-policies called integrity rules. Although it supports role-based access control, ASL struggles with scalability to larger systems and doesn't support grouping rules for reusability or explicit specifications for delegation.

- **ISPS (IPsec Security Policy Specification):** Applies policy constraints to entities such as security gateways and router filters. While it specifies confidentiality and integrity rules, it lacks features such as authentication, auditing, and delegation capabilities.

- **LaSCO (Language for Security Constraints on Objects):** Attempts to express constraints on objects using logical expressions and directed graphs, but does not directly address confidentiality, integrity, or support for delegation in grid environments.

- **PDL (Policy Description Language):** An event-based language that defines policies using an event-condition-action rule paradigm, mapping a series of events into a set of actions. It is unambiguous in semantics and supported by an enforcement architecture, but lacks support for access control policies and the composition of policy rules into roles or other structures.

B.30.3. What are the possibilities and challenges of automating the aspects of security policies in software systems?

Possibilities:

- **Dynamic Configuration:** Automation supports the dynamic configuration of systems to meet changing security requirements as dictated by specified policies. This dynamism is essential for adapting to new and evolving security requirements in various contexts such as authorization, access control, peer session security, quality of service, and network configuration.

- **Action Specification:** Tools and languages that specify how actions such as permissions and obligations can be expressed in an enforcement model help to address security needs more systematically.

Challenges:

- **High-level descriptions vs. implementation:** The gap between high-level descriptions of security policies and the concrete implementation details required by the policy enforcement architecture can be significant. Bridging this gap often requires manual intervention by programmers.
- **Platform-Specific Solutions:** Mapping policy specifications to actual implementations typically requires ad hoc, platform-specific solutions. These solutions may need to be developed from scratch each time, and may be difficult to reuse in different contexts.
- **Lack of common infrastructure:** There's a lack of a common infrastructure that can specify, monitor, and enforce security policies across a wide range of systems and applications.
- **Complexity of Specification:** Security policy specification itself can be a complex task due to the abstract nature of policy descriptions and the technicalities involved in specifying enforcement measures.

B.31. A Policy-Based Security Architecture for Software-Defined Networks [Var+19]

B.31.1. What are the key aspects of security policies in software systems, and how are they structured?

- **Policy Language Syntax and Wildcards:** The policy language uses wildcards in its syntax, allowing the specification of policies that can apply to sets or groups of entities and services. Policy expressions, when satisfied, trigger associated actions such as allowing or denying requests.
- **Protection and sharing policies:** The language facilitates the specification of protection policies that take into account the attributes of devices through which data may flow. For sensitive information, the paths and devices must meet certain security attributes. In addition, release policies can be specified for endpoints, requiring them to meet certain security attributes before traffic can be released.
- **Specifying paths and conditions:** Policies act as rules that specify specific paths that packets must follow within the network and the conditions under which packets must follow those paths.
- **Policy Repository and Specification:** A critical component of the security architecture is the specification of security policies to be enforced on SDN communications. These policies are stored in a policy repository. The approach to policy specification is language-based, inspired by RFC1102 for policy-based routing syntax.

- **Policy Expression Template:** A simplified policy expression template consists of attributes such as flow ID, autonomous system (AS) identifiers, host IP addresses, user information, domain constraints, services, security profile, and path, along with prescribed actions. This template allows the articulation of different policies for different users and domains within the SDN Policy Repository.

- **Policy Evaluation and Enforcement:** When network traffic matches a policy expression stored in the Policy Repository, the Policy Evaluation Engine instructs the Policy Manager to enforce the specified rules through the Policy Enforcer. This ensures that actions, such as allowing traffic according to policy, are performed on the network.

- **Fine Grained Policy Enforcement:** Security policies can be enforced at a fine-grained level, taking into account the context of flows, including location, routing information, services accessed, and security labels of switches and controllers, helping to detect and respond to threats in an SDN environment.

B.31.2. What are the possibilities and challenges of automating the aspects of security policies in software systems?

Possibilities:

- **Fine-Grained Security Policies:** The architecture allows the specification of granular security policies based on various attributes such as user and device parameters, context information, and switch and controller security attributes. This granularity ensures that policies can be dynamically configured to meet specific security requirements.

- **Detect and Respond to Attacks:** Automatic enforcement of security policies across multiple domains helps detect and prevent threats such as spoofing and flooding, ensuring that only secure and authorized traffic flows between domains.

- **Dynamic Configuration:** Dynamic security policy configuration facilitates efficient response to changing threat landscapes and can counter different types of attack streams to secure the delivery of SDN services.

- **Distribution of Security Capabilities:** The architecture enables intelligent distribution of security capabilities across the network, increasing system agility and responsiveness to security threats.

Challenges:

- **Complexity of Policy Management:** As the granularity and dynamics of security policies increase, management complexity can increase, requiring advanced tools and expertise.

- **Performance Overhead:** The need to evaluate and enforce complex policies in real time can potentially impact network and service performance, challenging the scalability of security automation.
- **Interoperability Issues:** In multi-domain environments with different SDN controllers, achieving seamless policy enforcement to accommodate the different policies and protocols of each domain can be problematic.
- **Security of Automation Tools:** The tools and infrastructure used to automate security policies must themselves be secure. There is a risk that if these tools are compromised, the integrity of the security policies could be undermined.
- **Adaptability to Emerging Threats:** Automated systems must quickly adapt to new threats, requiring continuous updates to policy rules and robust threat detection and response mechanisms.
- **Compliance and Regulatory Requirements:** Automating security policies means that they must also comply with relevant laws and regulations, which can change and vary from jurisdiction to jurisdiction.

B.32. Brew A Security Policy Analysis Framework for Distributed SDN-Based Cloud Environments [Pis+19]

B.32.1. What are the key aspects of security policies in software systems, and how are they structured?

- **Classification of Conflicts:** This paper discusses the classification of potential conflicts in a cloud environment, with a particular focus on cross-layer conflicts, and the development of controller-based algorithms to detect and resolve intra- and inter-table flow rule conflicts.
- **Methodology:** The methodology involves extracting flow rules in a distributed controller environment and includes cross-layer conflict checking to help detect conflicts.
- **Conflict Resolution:** The framework provides both automatic and assisted conflict resolution mechanisms, as well as a visualization scheme to graphically represent potential conflicts.
- **Global Prioritization:** Brew includes techniques for global prioritization of flow rules depending on the decentralization strategy.
- **Extended firewall rule conflict classification:** The framework extends firewall rule conflict classification to SDN flow rule conflicts by identifying cross-layer conflicts. It also introduces mechanisms to resolve these conflicts without administrator intervention.

- **Visualization:** The framework presents a novel visualization scheme to help administrators graphically display flow rule conflicts.

B.32.2. What are the possibilities and challenges of automating the aspects of security policies in software systems?

Possibilities:

- **Automatic Conflict Resolution:** The Brew framework provides strategies for automatically resolving flow rule conflicts, making it possible to maintain a conflict-free environment without constant human intervention. This automation increases security confidence and packet processing efficiency.
- **Performance Scalability:** The conflict detection and resolution algorithms designed into the method grow linearly with the number of flow rules, indicating scalability even for large SDN-based clouds.

Challenges:

- **Complexity in highly dynamic environments:** The current conflict resolution model in Brew may not be suitable for highly dynamic environments because it does not fully account for the dynamic nature of address mappings in such environments.
- **Topology Consideration:** Current conflict detection does not explicitly take network topology into account, which may limit the thoroughness of conflict detection. Inclusion of topology information may allow for more comprehensive analysis.
- **Linear Processing Time:** While the runtime complexity of the framework is linear and scalable, there is room for optimization. The current processing time grows linearly with the number of flow rules, and optimization of data structures could potentially reduce it further.
- **Policy Implementation Centralization:** There is an implicit recognition that centralizing policy implementation is critical to effective security, which could be a challenge if local flow rules are given higher priority without regard to the overall security posture.

B.33. Intuitive Security Policy Configuration in Mobile Devices Using Context Profiling [Gup+12]

B.33.1. How can the aspects of security policies in software systems be tailored towards the concerns of stakeholders?

- **User-centric design:** The framework is designed with the end user in mind, seeking a balance between usability and security. Instead of using a one-size-fits-all

approach or complicated manual configurations, the system personalizes security settings based on the user's actual environment and behavior.

- **Context Sensitivity:** Using the various sensors available on modern smartphones, the system gains an understanding of the environment. This context sensitivity allows the system to adapt security policies that take into account the different needs of stakeholders in different situations, such as different locations or levels of environmental risk.

- **Automated Security Decisions:** Policies are dynamically adjusted, reducing the burden on users who may not have the expertise or desire to manually adjust security settings. For example, the lockout timeout and unlock method for a device can be automatically configured based on the perceived security of the current context.

- **Device and Context Familiarity:** Security policies are tailored based on a familiarity score, which is a representation of how often a user or their device encounters specific Bluetooth or WiFi devices. This provides a more personalized security setting that is relevant to a user's regular activities and movements.

- **Privacy:** User privacy is a fundamental design principle of the framework. All contextual data is stored and processed locally on the user's device, addressing stakeholder concerns about privacy and data protection.

- **Stakeholder Feedback:** The framework allows for stakeholder input in the form of feedback on the perceived security of a context. This feedback can be used to quickly adjust to the correct security levels inferred by the context profiler and to accommodate individual user perceptions of security, thus providing a mechanism for stakeholder involvement in policy personalization.

B.33.2. What are the key aspects of security policies in software systems, and how are they structured?

- **Configuring Access Control Policies:** Configuring access control policies can be complex and overwhelming for users. The framework attempts to automate this process by using sensed environmental information such as Bluetooth devices, WiFi access points, temperature, ambient light, sound, and location coordinates. This information allows the system to infer the familiarity and security of a context, which can aid in access control decisions.

- **Context Profiling Framework:** The proposed framework profiles contexts to infer reasonable access policies without user intervention. A device periodically scans its environment and profiles Contexts of Interest (CoIs) by tracking encounters with WiFi and Bluetooth devices. Profiling incorporates user feedback on perceived security.

B.33.3. What are the possibilities and challenges of automating the aspects of security policies in software systems?

Possibilities:

- **Context-Aware Automation:** The framework proposes the use of environmental sensors on mobile devices to automatically infer context familiarity and device security. By collecting data such as GPS locations, WiFi networks, and Bluetooth device presence, security policies can be dynamically adjusted based on the user's context, enhancing protection without user intervention.
- **Improved User Experience:** Automated security policies provide a more seamless user experience by eliminating the need for users to manually configure complex settings, which can be both tedious and beyond the expertise of many users. The system's ability to contextually adapt security measures, such as device locking mechanisms, removes this burden from the user.
- **Enhanced Privacy and Security:** By processing all contextual data locally on the device, the framework enhances the privacy and security of user data. No raw sensor data or derived contextual parameters need to be sent outside the user's device, reducing the risk of data breaches and unauthorized access.

Challenges:

- **Context Profiling Accuracy:** Creating an accurate profile of a user's context that can reliably infer security and adjust security policies accordingly is challenging. It involves complex data models and algorithms that must correctly interpret sensor readings. There is a need for continuous improvement and validation of these models to ensure accuracy in different situations.
- **Balancing security and convenience:** It's important to maintain a balance between strong security measures and not overly inconveniencing the user. This can be challenging, as there may be situations where the system misinterprets the context, leading to security measures that are either too lax or too restrictive.
- **Integration of User Feedback:** Incorporating corrective user feedback into the security policy without compromising automation can be challenging. The system must incorporate feedback accurately and adaptively, possibly using machine learning models that can be computationally intensive and require constant tuning.
- **Handling absent or erroneous sensor data:** Situations where GPS is unavailable or sensor data is erroneous (such as indoor environments) require robust fallback mechanisms. The system must infer context using other local mechanisms, which is a challenge to maintain accuracy without server-assisted positioning.
- **Mapping Familiarity to Security:** The interpretation of familiarity in terms of security varies by user and application. Developing a one-size-fits-all mapping

strategy may not be feasible, so the system must incorporate flexible models that can accommodate different circumstances and stakeholder preferences.

- **Privacy vs. Responsiveness:** Ensuring that all data processing takes place locally to maintain privacy can conflict with the need for responsiveness and the use of cloud-based resources to enrich context determination. Finding the right balance remains a challenge that requires innovative solutions in edge computing and local computing techniques.

B.34. Extending the Java Virtual Machine to Enforce Fine-Grained Security Policies in Mobile Devices [IDC07]

B.34.1. How can the aspects of security policies in software systems be tailored towards the concerns of stakeholders?

- **Per-Application Policies:** This paper describes the need to support fine-grained, history-based, and user-definable per-application policy specification and enforcement within the J2ME framework. For example, stakeholders may require a policy that a browser cannot download more than a certain amount of data per day.

- **Policy Manager:** To meet the varying needs of stakeholders, each MIDlet (a type of mobile application in J2ME) can be associated with a specific policy that becomes part of the corresponding MIDlet suite. The Policy Manager is responsible for managing and interpreting both MIDlet-specific and system-wide policies.

- **User Configuration:** A user interface can be implemented to allow users to set desired policies on their mobile phones. This allows stakeholders to actively participate in security policy configuration according to their individual needs and preferences, thereby increasing stakeholder buy-in and compliance with the security policy.

- **History Keeper:** This module would maintain a history of resource access and usage, allowing policies to be evaluated against past system and application behavior. This is especially important for stakeholders who need to ensure compliance over time, not just at a single point in time.

- **Security Policy Language (SPL):** The paper suggests using SPL to express flexible and dynamic security policies that support access control and can be history-based. SPL allows the specification of rules that guide system behavior on a per-application basis, with the ability to prioritize policies. This provides the granularity that stakeholders need to effectively manage their unique security concerns.

- **Global System Policy:** While local policies address the security needs of individual applications, a global system policy sets the cumulative limits on resource

consumption for all applications. This broader platform policy provides an overarching structure within which stakeholders can define their localized policies.

B.34.2. What are the key aspects of security policies in software systems, and how are they structured?

- **Sandbox Model:** This is a fundamental concept in the general Java security architecture. A sandbox is an execution environment with strict policy-based resource access control and strong isolation properties. Each sandbox is associated with a protection domain that defines the permissions granted to the application running within it.

- **Protection Domains:** The Java Virtual Machines (JVMs) allow protection domains to be defined through security policy files. The permissions defined in these policy files are dynamically assigned at runtime. This model distinguishes between trusted applications, which are generally allowed to run unrestricted, and untrusted applications (such as applets and remote code), which are subject to the security policy.

- **Configurability:** For the Java 2 Standard Edition (J2SE), the Java Security Policy is fully configurable externally by platform users and administrators. They have the freedom to define both the permissions and the domains.

- **Trust Model:** The trust model classifies applications as trusted or untrusted. A trusted application is one that is generally allowed to run with fewer restrictions, while untrusted applications are more restricted.

- **Simplified Security in J2ME:** Because of the limitations in the capabilities of devices running Java 2 Micro Edition (J2ME), the security architecture is greatly simplified. While this simplification addresses resource consumption issues, it comes at the cost of flexibility and granularity of policy specification and enforcement.

- **Permission Categories in MIDP:** Within the domain-based security model maintained by the Mobile Information Device Profile (MIDP), permissions are classified into groups of "ALLOWED" permissions, which are granted automatically, and "USER" permissions, which are granted upon explicit user approval.

- **Operational Aspects:** In the original J2ME security architecture, applications running on the CLDC and MIDP run inside the KVM, a lightweight version of the JVM. The sandbox model defined by KVM differs in that it limits the exposed APIs, restricts application management to the native code level, and does not allow users to modify classloaders or download native libraries.

B.35. Security policy checking in distributed SDN based clouds [PCH16]

B.35.1. What are the key aspects of security policies in software systems, and how are they structured?

- **Centralized Security Policy Management:** SDN enables centralized implementation and management of security policies in a cloud environment by separating network control from network devices. This centralization leads to more efficient management of security protocols, as dynamic network topology changes or host reconfigurations can be addressed quickly and comprehensively.

- **Complexities of Security Implementation:** The flexibility and programmability of SDN enables rapid response to changing user and security requirements through dynamic network reconfiguration. However, this same flexibility introduces complex issues unique to SDN, such as flow rule chaining, cross-layer policy conflicts, partial matches, and set-field action challenges.

- **Flow Rule Chaining and Conflicts:** The presence of flow rules from different users on a shared control plane can lead to potential flow rule conflicts. In SDN environments, any controller module can introduce new flow rules that may conflict with existing rules or the security policy itself. Unlike traditional networks, where an administrator adds rules within a hierarchical structure, SDNs operate in a more distributed manner, which can make conflict resolution more difficult.

- **Cross-Layer Policy Conflicts:** This paper introduces a new class of conflicts, called "imbrication," which accounts for indirect cross-layer policy conflicts. This phenomenon is exacerbated in SDN environments because policy enforcement points on different layers (layer 2 or layer 3) may enforce conflicting policies, resulting in inconsistent actions.

- **Conflict Detection and Resolution Framework:** The authors of the study present a framework that uses a formalism for reasoning about OpenFlow rule conflicts in dynamically changing cloud environments. This includes a controller-based algorithm for detecting conflicts in flow tables and mechanisms for automatic and assisted conflict resolution. The framework also addresses scenarios where not all parties in a multi-tenant cloud may cooperate, ensuring fair enforcement of security policies.

- **Related Work and Approaches:** The paper references several previous works on security solutions in SDN, such as the basic SDN firewall introduced as part of Floodlight, layer-2 firewalls, and security services such as Pyretic and FRESCO. These works have provided different ways of handling security rules in an SDN context, each with their own approaches to conflict management and rule enforcement.

B.35.2. What are the possibilities and challenges of automating the aspects of security policies in software systems?

Possibilities:

- **Centralized Management:** SDN's centralized architecture simplifies the automation of security policy implementation and management across entire network environments.
- **Dynamic Adaptation:** Automated systems can quickly respond to security requirements and changes, such as dynamic network configurations, to update flow rules in real time to align with the organization's security policy.
- **Real-Time Detection:** With SDN, security infrastructures can detect changes in real time and update flow rules on distributed controllers accordingly, ensuring consistent enforcement of security policies.
- **Conflict Resolution:** This paper presents an algorithm and framework for automatically detecting and resolving flow rule conflicts, including indirect cross-layer policy conflicts, and includes automatic and assisted resolution mechanisms.

Challenges:

- **Complexity of Flow Rules:** SDN environments can have more complex security policies due to their ability to tune at multiple layers (beyond layer 3 and layer 4 headers). Automating the enforcement of such rules can be challenging due to the number of variables and possible actions such as forwarding, packet modification, and rate limiting.
- **Cross-Layer Policy Conflicts:** Specialized tools are required to address the heightened problem of cross-layer policy conflicts in distributed environments such as SDN. Automatic handling of this problem requires consideration of cross-layer dependencies to ensure conflict-free policies.
- **Role-Based and Attribute-Based Policy Conflicts:** Including role-based and attribute-based conflicts is a future area of research mentioned in the paper, which means that automating policies with respect to user roles and attributes is a recognized challenge that has not yet been fully addressed.
- **Diverse Environments:** The paper suggests that automating security policies in environments with multiple diverse controllers is still a challenge that requires further development of the proposed framework.
- **One-size-fits-all solution:** Since a one-size-fits-all solution is unlikely to work for every scenario, it is necessary to explore different variations of the framework tailored for specific types of environments, such as host-based SDN firewalls or mobile SDN clouds.

B.36. Security policy enforcement in the OSGi framework using aspect-oriented programming [PS08]

B.36.1. What are the key aspects of security policies in software systems, and how are they structured?

- **Suppression:** Policies that prohibit an action by ignoring it, suitable for non-critical actions. An example is suppressing an alert message when the speed of a vehicle exceeds 80 mph.
- **Insertion:** Policies that allow an action, but require additional code to be executed before or after the main action. For example, saving a bundle object before it starts in the OSGi platform.
- **Truncation:** Policies corresponding to the runtime monitor approach that abort execution if a prohibited action is attempted.

B.36.2. What are the possibilities and challenges of automating the aspects of security policies in software systems?

Possibilities:

- Implementation of the reference monitor approach, in which a target program is modified to comply with a security policy when executed by adding new code for security checks.
- The use of expressive languages and tools such as PSLang/PoET for enforcing security policies in Java bytecode, and Polymer for more expressive security policies in Java applications that provide powerful transformational responses to application events.
- AspectJ, an aspect-oriented programming language, can be used to implement policy enforcement that can be applied at the Java bytecode level.
- Aspect-Oriented Programming (AOP), particularly with AspectJ, allows for the modularization of cross-cutting functionality so that security policy enforcement can be accomplished by defining pointcuts for security-relevant events and corresponding advice for security responses.

Challenges:

- Current tools for program transformation that enforce security policies are research prototypes and lack the robustness and completeness of mature industrial tools, limiting experimental investigation in real-world contexts.
- It remains to be seen whether AspectJ or similar aspect languages can be scaled up to real systems as effective policy languages.

- Various aspects, such as the maturity of the security guarantees provided by AspectJ, need to be investigated.
- In certain frameworks, such as the OSGi framework used in automotive systems, existing security mechanisms may have limitations. For example, the Java 2 security model used in OSGi cannot define policies based on execution history or runtime variable values other than what is visible through stack inspection.
- Proposals such as rule-based runtime monitors built into the OSGi framework, while relevant, have potentially high runtime costs.
- Until this research, no previous studies have comprehensively addressed security policy enforcement for OSGi using aspect-oriented programming, making this a new and relatively unexplored area.

B.37. Owned policies for information security [CC04]

B.37.1. What are the key aspects of security policies in software systems, and how are they structured?

- **Information Flow:** Static information flow control enforces end-to-end security properties, in particular noninterference, which ensures that low-security outputs do not depend on high-security inputs. In systems with mutually distrustful principals, each principal may have different information flow requirements, and when their information is combined, each principal's restrictions must be applied. One model here is the decentralized label model, which serves as a model for proprietary information flow policies
- **Access Control:** In systems with information resources such as files owned by principals, owners can specify access control lists that define who can access their resources. For example, a procedure called by a principal that accesses multiple resources should have access controls that are as restrictive as the combined access controls of the resources it accesses. Owner Retained Access Control is an example of a model for owner access control policies
- **Software Licensing:** Software vendors distribute software components under licenses that place restrictions on how and when the software can be combined with other components to form new software. The GNU General Public License is an example, where distributed software must also be under the GPL when used as part of other software. The owners here are the producers, and the guidelines restrict the composition of software components.

B.37.2. What are the possibilities and challenges of automating the aspects of security policies in software systems?

Possibilities:

- **Reducing the Specification Burden:** Security policy inference can reduce the burden on developers to write extensive security policy specifications, which can be especially significant for large systems.
- **Easing Rapid Development:** By automating policy specification, developers can more quickly develop secure systems without having to manually annotate every aspect of the security policy.
- **Reducing Annotation Clutter:** Automation can help minimize the "clutter" of annotations in programs, potentially making the code base easier to understand and maintain.
- **Framework for Custom Policies:** The framework provided is conducive to security policy inference, and the authors of the paper have demonstrated positive tractability results for policy inference within the framework, indicating that automation of this inference process can be reliable and efficient.

Challenges:

- **Handling Partial Knowledge:** One of the biggest challenges is automating security policy inference and enforcement when only partial knowledge of the runtime security policy structure is available. This situation is common because aspects of a system, such as user roles or access permissions, may change frequently.
- **Inference complexity:** Security policy inference, especially with partial knowledge, can be complex. Methods must be robust enough to deal with the uncertainty and dynamic changes within a system, while still maintaining security guarantees.
- **Integration with Existing Systems:** Automation must be compatible with existing systems and infrastructure, requiring careful design and potentially complex integration efforts.
- **Maintain Accuracy and Completeness:** Automated systems must ensure that inferred policies are accurate and complete, and adequately represent the intentions of system administrators and policy owners.

B.38. Towards an automated firewall security policies validation process [AG08]

B.38.1. What are the key aspects of security policies in software systems, and how are they structured?

- **Subject:** Represents an active entity in the system, such as human users, employees, processes, applications.

- **Object:** Represents a passive entity in the system (also called an asset or resource), such as ports, data, or hosts.
- **Operation:** Represents an action that can be performed by a subject on an object, such as a connection or read/write requests.
- **Security Rule:** Expresses the appropriate security decisions (allow or deny) to be made for each operation attempt.

B.39. Semantic remote attestation for security policy [ZHM10]

B.39.1. What are the key aspects of security policies in software systems, and how are they structured?

- **Security Policy Mutability:** Security policies are more mutable and dynamic than binary codes. These policies contain a set of rules that can change, causing the binary expression of the policy to be infinite. This makes traditional integrity measures such as hash values difficult to verify for security policies.
- **Mixture of Semantics:** Security policies are often implemented as a single policy file that controls an entire platform. This file mixes many different semantics, and although only a small subset of these semantics may be required in a real-world scenario, the presence of unrecognizable or untrustworthy semantics renders the entire policy hash untrustworthy.
- **Certifying Trustworthiness:** Traditional approaches to proving trustworthiness by measuring integrity are not sufficient for security policies. Instead, there is a need for other types of evidence to express the trustworthiness of security policies. Two main types of evidence are discussed: integrity measurements (e.g., hash values of objects) and security properties of the object. The paper argues for the use of security properties, or semantics, over integrity measurements.
- **Novel Semantic Approach for Attestation:** The paper proposes a novel semantic approach for attesting the trustworthiness of security policies. This approach involves using policy analysis techniques to extract semantics from security policies, and logical programming to query the necessary semantics. This approach avoids problems associated with hash values and increases the flexibility and applicability of attestation methods.
- **Security Policy Structure:** The paper does not provide explicit details on the structural elements of security policies within the quoted sections. However, it is suggested that traditional security policy files are structured to govern the rules of the entire platform, which poses challenges for the separation and specific attestation of certain semantics needed in a given context.

B.40. Policy Components - A Conceptual Model for Tailoring Information Security Policies [RKG22]

B.40.1. What are the key aspects of security policies in software systems, and how are they structured?

Design: Each policy component within an Information Security Policy (ISP) is designed to be self-contained, expressing rules that guide and protect an organization's assets while performing specific tasks. This approach is intended to align with the objectives relevant to the stakeholder's role and responsibilities within the organization. Alignment: The goals of policy components are explicitly stated to address potential goal conflicts and provide a clear direction for stakeholders to work toward, thereby aligning with their interests and concerns.Different policy statements for different purposes: Policy statements within an ISP are categorized as actionable advice, educational content, and general content. Actionable advice provides clear instructions and rules for performing tasks, educational content raises awareness and educates about information security, and general content informs users about security policies. These serve different communication objectives and stakeholder needs. Clarity: The policy allows for the definition of key concepts to ensure a common understanding among all users. This is important because information security and business terminology can be complex, and stakeholders need a clear understanding of such terms in order to comply with the policy. to Supplementary Sources: The policy components contain references to supplementary sources such as laws, regulations, and standards. This ensures that the advice given is not only consistent with legal requirements, but also informed by best practices and standards that are important to stakeholders. for non-compliance: By including specific consequences for non-compliance with the actionable advice, the policy component supports deterrence and clarifies accountability, which is critical for stakeholders concerned with compliance and organizational discipline. Presentation: The structure of the ISP organizes related policy components in a way that provides a clear overview for stakeholders in their respective roles. This satisfies the requirement for clarity and ensures that stakeholders find relevance and order in the policy according to their concerns. Tailored ISPs: Policy components act as building blocks to construct tailored ISPs that are designed for specific roles and tasks. This ensures that stakeholders have guidance that is directly relevant to their roles within the organization.

B.40.2. How can the aspects of security policies in software systems be tailored towards the concerns of stakeholders?

and Role: The model begins with actors (individuals associated with an organization) and roles (functions played by an actor within an organization). This division ensures that the ISP addresses clear and consistent audiences. Policy Components:

Tailored to guide and restrict tasks associated with specific roles, policy components are self-contained parts of ISPs designed to protect organizational assets. They are driven by objectives, verifiable states of the world that the policy seeks to achieve, often derived from internal standards, laws, and regulations. Statements: There are three different types of policy statements-actionable, educational, and general-each of which serves a different communication purpose. Actionable statements provide specific instructions and rules for performing tasks, while educational and general statements provide information intended to educate or inform users without regulating behavior. : The policy includes consequences, which are specific sanctions for noncompliance, to deter noncompliant behavior.Concepts: To ensure common understanding, the model allows for the definition of concepts within the organization that are associated with the various pieces of advice. Supplementary Sources: These are artifacts that provide additional background information for the advice offered within the policy components, ensuring that they are informed by relevant laws, regulations and standards. Structure: The structure of an ISP is critical to organizing related policy components in a way that provides an overview to the actor for a given role, thus addressing the need for clear structure within security policy design.

B.41. Security Policy Modelling in the Mobile Agent System [H19]

B.41.1. How can the aspects of security policies in software systems be tailored towards the concerns of stakeholders?

The aspects of security policies in software systems can be tailored to the concerns of stakeholders by using generalized security ontologies. These ontologies are designed to cover security features that form an explicit domain terminology for stakeholders with different interests and perspectives. By contributing to a shared knowledge base, these ontologies support security development and provide a common logical understanding that can be used without human intervention. This implies that stakeholders' security concerns can be addressed by incorporating their specific requirements and capabilities into the security ontologies, which then inform the development and implementation of the respective security policies.

B.41.2. What are the key aspects of security policies in software systems, and how are they structured?

Capabilities:

- Security policies are designed to protect both the resources and data of the host system and the integrity and confidentiality of the mobile agents and their communications.

- The paper emphasizes the importance of ensuring interoperability and resolving heterogeneity between security policies by integrating semantics through security ontologies that provide a common knowledge base.The paper emphasizes the importance of ensuring interoperability and resolving heterogeneity between security policies by integrating semantics through security ontologies that provide a common knowledge base.

- Security requirements allow the specification of the necessary security settings for the secure execution of a mobile agent.

- Security capabilities are features provided by mobile agents or platforms that allow tasks to be executed with a certain level of security.

B.41.3. What are the possibilities and challenges of automating the aspects of security policies in software systems?

Possibilities:

- Automation facilitates the analysis of semantic compatibility between security policies by considering not only the concepts, but also the properties of those concepts, which is critical due to the extensive use of property attributes in security annotations.

- The iterative creation process of these ontologies allows for continuous refinement and expansion to accommodate new security statements as they are evaluated and contributed to by the security community.

Challenges:

- The creation of ontologies is an iterative process, and additional instances and properties will continually be needed to express new security claims.

- There is a need for continuous adaptation, where classes and properties are added or deleted based on the evolving consensus within the security community.

- Ontologies will need to evolve to address broad issues such as access control policies and quality of service.

- Ontologies are currently used in specific contexts, such as communication scenarios between mobile agents and platforms, and may not be fully equipped to handle the heterogeneity and specificity of security policies required in different areas of the mobile agent system.

B.42. Specification of information flow security policies in model-based systems engineering [Ger18]

B.42.1. How can the aspects of security policies in software systems be tailored towards the concerns of stakeholders?

- By extending structural systems engineering models to include flow policies, the paper enables the specification of security requirements against unauthorized information flows that need to be prevented.
- Through structural decomposition, the paper describes the refinement of specified flow policies into more fine-grained security requirements.
- It also shows the translation of refined flow policies into security requirements for individual software components.

B.42.2. What are the key aspects of security policies in software systems, and how are they structured?

- The technique of specifying information flow security policies within the system development process.
- The refinement of the specified policies to derive fine-grained security requirements at the subsystem level.
- The theory of information flow security is used to detect information leaks in the information processing of a software system.
- The information processed by a system is categorized into different security domains, and the security policy restricts the flow of critical information from one domain to certain other domains.

B.42.3. What are the possibilities and challenges of automating the aspects of security policies in software systems?

The opportunities and challenges of automating security policies in software systems, as discussed in the paper, include

Possibilities: The formal semantics of information processing allows automated verification of component security policies. This means that for individual software components, security policies can be automatically verified using established methodologies from previous work, representing a significant advance in the automation of security enforcement in software systems.

Challenges: A key challenge identified is that the information flow security of individual components is not necessarily preserved when those components are assembled. For example, even if two components, such as engine control and storage gateway, comply with their individual security policies, there may still be an unintended flow of information between other components, such as predictive maintenance and user interface, which could lead to security vulnerabilities. Addressing this issue is critical to maintaining the overall security of the software system once individual components are integrated.

B.43. Automated legal compliance checking by security policy analysis [RS17]

B.43.1. How can the aspects of security policies in software systems be tailored towards the concerns of stakeholders?

- **Collaboration between legal and IT security experts:** Legal and IT security experts work together to identify parts of policies that are amenable to formalization, make explicit any simplifying assumptions, and derive a mathematical model that bridges the gap between technical and legal understanding.
- **Production of Guidelines:** A set of natural language guidelines is produced to assist IT system designers in aligning technical design with legal requirements.
- **Design of Access Control Policies:** The IT security experts design access control policies that reflect the key processes in the system, ensuring that each process serves a specific purpose.
- **Specification of Bridge Structures:** Experts specify a bridge structure to instantiate the formal model of regulatory requirements into a policy applicable to the system under design.
- **Using Automated Policy Analysis Tools:** Such tools can be used to check the conformance of the security policy with the formalized legal requirements. They can answer questions about the security of the system design and verify that the system meets the required legal standards.
- **Scenarios Explaining Violations:** The tools can provide scenarios that explain why the legal requirements are violated when a violation is detected, which can guide the modification of the system design or policy to address stakeholder concerns.

B.43.2. What are the possibilities and challenges of automating the aspects of security policies in software systems?

Possibilities:

- **Security-by-Design:** Incorporating security from the design phase into every part of the IT management process.

- **Automated Techniques:** Several automated techniques are available for verifying security policies, especially those related to access control requirements, including scenario discovery and change impact analysis.
- **Regulatory Compliance:** Security policy auditing tools can help ensure regulatory compliance, which is becoming increasingly important with data protection and privacy laws such as the EU Data Protection Directive.
- **Ongoing Compliance:** For systems with frequent code updates or scalable infrastructure, such as cloud-based systems, automated techniques are the best hope for ensuring continuous security policy and regulatory compliance.

Challenges:

- **Alignment with Regulatory Requirements:** The key challenge is to ensure that evolving security policies remain compliant over time.This is not just a matter of deploying technical security mechanisms, but also of maintaining that compliance.
- **Frequency of System Updates:** Today's software systems, especially those deployed in the cloud, are updated frequently, which increases the difficulty of maintaining compliance.
- **Infrastructure Scalability:** Because infrastructure is designed to be scalable, security policies must be adaptable to handle that scalability while maintaining compliance.
- **Need for Compliance-by-Design:** Continuous compliance relies heavily on a compliance-by-design approach supported by automated techniques and tools that can adapt to policy evolution.

B.44. Security policy scheme for an efficient security architecture in software-defined networking [LK17]

B.44.1. What are the key aspects of security policies in software systems, and how are they structured?

- **Separate:** Divides the virtual services and reduces the size of attack flows using the load balancer.
- **Connecting:** Links many Virtual Network Functions (VNFs) to prevent different attack flows and build large security systems.
- **Merge:** Combines unnecessary VNFs to optimize the security system and system resources.
- **Reorder:** Rearranges current VNFs based on the type and strength of current attack streams.

B.45. Closing the gap between the specification and enforcement of security policies [HPF14]

B.45.1. How can the aspects of security policies in software systems be tailored towards the concerns of stakeholders?

- **Modeling Security Functionalities:** First, all the possible features of each security functionality are specified in a tree-based diagram that forms the Software Product Line (SPL). This variability model is then linked to the complete implementation of the security functionality. Each feature in the diagram is linked to specific pieces of code that must be included and configured to support the desired functionality. This linkage can include classes, parameters, or values.

- **Create configuration per security policy:** For each security policy, a configuration of the previously specified SPL is created by selecting features that satisfy the requirements of the security policy, effectively instantiating the SPL. This selection can be done manually by a security expert, or automatically using a reasoning engine that analyzes the security policy and selects the appropriate features to satisfy stakeholder concerns.

B.45.2. What are the key aspects of security policies in software systems, and how are they structured?

- **Security Policy Definition:** A security policy is a set of rules that govern the nature and context of system actions according to specific roles, such as permissions, prohibitions, obligations, availability, etc., with the goal of ensuring and enforcing security.

- **Specification and Enforcement:** Security policies must be specified before they can be enforced, typically using models such as OrBAC, RBAC, MAC, etc. to describe the security properties that applications should meet. Enforcement involves providing certain security functionality within the application to support the policy

- **Variable Deployment:** Applications can be deployed with different security policies, which means that a variable number of security features can be used, but not all features are deployed at the same time. Secure deployment of an application is challenging because of the gap between security policy specification and enforcement.

- **Software Product Lines (SPLs):** SPLs are used to model the commonality and variability of security properties in security policies, associate security properties with modules that implement security functionality, and automatically generate a security configuration that includes only the security functionality required to enforce a particular security policy.

- **Aspect-Oriented Programming (AOP):** AOP is used to design and implement security functionality separately from applications, as aspects, allowing a security configuration to be deployed in an application without modifying the original application.

- Modeling commonality and variability of security properties at an abstract level to enable different configurations for implemented functionality, ensuring that a wide variety of applications can be deployed quickly, comprehensively, and consistently.

- Generate security configurations for different levels of security required by different security policies, enabling customized deployments for the same application using SPL.

- Runtime adaptation to enforce changing security policies, as is being done in projects such as the European Inter-operable Trust Assurance Infrastructure (INTER-TRUST), which supports trusted applications that adapt to changing security requirements.

B.45.3. What are the possibilities and challenges of automating the aspects of security policies in software systems?

Possibilities:

- **Correctness:** Software Product Lines (SPLs) and Aspect-Oriented Programming (AOP) do not inherently improve the correctness of applications or security features. However, the modularization of security concerns with AOP helps to facilitate the verification of security properties in applications, since security experts only need to check the code of the aspects and the definition of the pointcuts where the aspects are introduced, rather than checking all the modules in the base application.

- **Maintainability and Extensibility:** SPLs allow for easy adaptation to changes in security policy specifications by reconfiguring the security functionality to accommodate those changes. The variability model can also be extended to cover additional security concerns, enhancing the system's ability to evolve and incorporate new requirements or policy changes.

- **Reusability:** The approach allows security functionality to be reused across different applications and security policies. However, the generated aspects are specific to particular security policies and may contain application-dependent knowledge, such as pointcuts, which limits the reusability of the exact aspects across different applications.

- **Scalability:** Variability models can lack scalability because the tree diagrams representing the models become unwieldy as they grow. However, CVL (Common Variability Language) can be used to manage complexity by decomposing the model into different levels of detail with features such as Composite Variability Features

and Configurable Units. This element helps to handle the variability aspects in a structured and more scalable way.

B.46. PoliSeer A tool for managing complex security policies [LL11]

B.46.1. What are the key aspects of security policies in software systems, and how are they structured?

- **Complexity:** Software security policies are often difficult to specify, understand, and update due to a variety of factors, including limited access to resources such as files, memory, and peripherals, auditing of security-related operations, and the use of cryptographic protocols.
- **Modularization:** The concept of modularizing complex software into simpler, reusable components such as packages, classes, functions, and aspects exists in software engineering, but this technique has limited application when it comes to modularizing complex security policies. Existing tools for this purpose may encapsulate entire policies as non-decomposable modules or allow for fine-grained policy modularization, but require expertise to use correctly.
- **Policy Composition:** Users who are not expert policy engineers can use PoliSeer, a GUI-based tool, to specify, visualize, modify, and enforce complex runtime policies on untrusted software. They do this by importing universally composable policy modules from a policy library and then composing them into more complex policies. These modules are typically predefined by expert policy engineers and allow for flexibility in policy composition.

B.47. Modelling mobility aspects of security policies [Har+05]

B.47.1. What are the key aspects of security policies in software systems, and how are they structured?

- **Abstract Representation of Security Policies:** This paper describes an abstract version of a security policy using a ping system and policy as an example. The policy is designed using C preprocessor macros to represent optional and multiple system calls, translating the language constructs into executable actions within the policy framework.
- **Least Privilege Principle:** The security policies are designed to follow the principle of least privilege. For example, the ping system specifies that the system should drop root privileges after a socket call, which demonstrates compliance with this principle. This principle ensures that the system has only the privileges it needs for a given task, limiting the scope and potential damage of an exploit.

- **Model Checking for Policy Conformance:** The paper illustrates how model checking can be used to validate that the behavior of the system conforms to the security policy. During this process, traces that violate the policy can be detected, helping to identify potential security problems. This aspect reflects the importance of defining clear criteria that security policies should satisfy, and using formal methods to verify compliance.
- **Separation of Concerns:** The paper points out that the separation of system and policy is a crucial structural aspect. For example, in a database application, the business logic layer is considered the policy because it determines acceptable behavior by controlling where data requests are routed (test or production environments). This separation provides clarity in understanding the different roles and responsibilities in managing and enforcing security within the system.

B.48. Security policy configuration issues in Grid computing environments [AGL03]

B.48.1. How can the aspects of security policies in software systems be tailored towards the concerns of stakeholders?

- **Delegation Management:** Security policies should specify which privileges can be delegated and which principals (including intermediaries) can receive them. More importantly, policies should protect delegated credentials by imposing controls such as validity and time expiration to prevent misuse and ensure that the right balance of rights is delegated to accomplish tasks without compromising security.
- **Identity Mapping:** For single sign-on capabilities, which are critical for stakeholder convenience, policies should facilitate the mapping of Grid identities to local user identities without violating existing access controls. This requires coordination between Grid and site administrators to establish trust relationships with Certificate Authorities (CAs), allowing CAs to manage the trust relationship between users and local domains rather than directly mapping ids.
- **Interoperability of Policies:** Security policies should integrate with local security solutions to maintain a high level of security without requiring changes to local resources. This is essential to accommodate inter-domain access and to ensure that stakeholder requirements for different domains are met.
- **Information Services Security:** Securely controlling access to Grid information services is critical for stakeholders concerned with resource location and availability queries. This includes establishing strong authentication, authorization, confidentiality, and integrity procedures for user interactions with these services. The use of protocols such as LDAP for authentication and authorization, along with the use of Grid credentials for access and X.509 credentials for confidentiality and integrity, can help secure this aspect of the policy.

B.48.2. What are the key aspects of security policies in software systems, and how are they structured?

- **Definition and Implementation:** The establishment of a secure Grid environment is critically dependent on the definition and implementation of a concrete security policy. Traditional security policy configuration, which typically involves access control lists associated with individual resources, is not sufficient for the complex needs of Grid environments, where multiple administrative domains and dynamic collections of resources and users coexist.
- **Dynamic Multi-User Environment:** Grid systems have a dynamic user population, with individuals having different local usernames, credentials, certificates, and accounts in different locations. Managing these changing identities and ensuring that different users are uniquely distinguished are key aspects of grid security policy.
- **Resource Pool Management:** Resources in grid environments are large and dynamic, with individual institutions determining their availability. Because resources can be combined in various ways for computation, security policy must address the dynamic nature of resource use and allocation.
- **Interoperability of Security Policies:** Security policy interoperability is critical, but also a complex issue in grid computing environments due to the involvement of multiple organizations with potentially different and conflicting security requirements. Policy integration must take into account different authentication and authorization mechanisms among the various security policies that coexist in a multi-policy environment.

B.48.3. What are the possibilities and challenges of automating the aspects of security policies in software systems?

- **Continuous update:** The vulnerability inventory should be continuously updated to reflect the latest knowledge and analysis of the security landscape.
- **Parameterization:** Upgrades and enhancements to the security policy should be parameterized to occur automatically without manual intervention. This would ensure that the system can adapt to new threats and changes in the environment without requiring constant human oversight.
- **Transparency:** Automated security policy changes should be transparent to users and participating entities. This means that the changes should be seamlessly integrated into the system without disrupting normal operations or requiring significant stakeholder interaction.
- **Effective remediation:** The ultimate goal of automation is to ensure that all introduced security problems are effectively addressed. This means that the automated system must be sophisticated enough to detect and mitigate a wide range of security problems as they occur.

B.49. Formal security policy verification of distributed component-structured software [Her03]

B.49.1. What are the key aspects of security policies in software systems, and how are they structured?

- **Interface Specifications:** Lists the methods, events, exceptions, and corresponding arguments of the component interfaces.
- **Behavioral Constraints:** This section describes the interface behavioral constraints that must be met by the component or its environment.
- **Synchronization Aspects:** The third part models the synchronization aspects between components.
- **Quality of Service Properties:** Finally, the contract lists the quality of service properties that must be guaranteed.

B.50. When role models have flaws Static validation of enterprise security policies [Pis+07b]

B.50.1. What are the key aspects of security policies in software systems, and how are they structured?

- **Identifying Required Roles:** RBAC policies must identify the roles that users need to run an enterprise application, ensuring that users have the permissions they need to perform their tasks unencumbered.
- **Detect Policy Inconsistencies:** It's important to detect potential inconsistencies caused by principal delegation policies that are used to override a user's role assignment and address any issues that may arise from the improper use of such delegation.
- **Policy Sufficiency Evaluation:** Policies should be evaluated for sufficiency. An insufficient RBAC policy can lead to stability problems due to potential runtime authorization failures if users do not have the necessary roles to access the required resources.
- **Redundancy Checks:** Policies should be checked for redundancy, meaning that users should not be granted more roles than are necessary for their tasks. This is consistent with the principle of least privilege, which states that users should have the minimum set of rights necessary to perform their jobs.
- **Prevention of subversive policies:** There is a need to prevent subversive policies that allow executions to bypass declared access restrictions, in particular by exploiting unchecked intra-component calls. By default, component-based systems

enforce authorization checks only across component boundaries. This behavior could allow security violations if intra-component calls are not properly monitored.

B.50.2. What are the possibilities and challenges of automating the aspects of security policies in software systems?

Possibilities:

- **Automatic Detection of Misconfigurations:** The use of formal models and static analysis models for RBAC policy validation enables security systems to automatically detect policy misconfigurations, such as insufficient, redundant, or subversive policies. This proactive detection can significantly improve system reliability and compliance with security standards.
- **Static Analysis Algorithms:** Algorithms have been developed to verify that an RBAC policy is sufficient and not subversive, and to identify redundant policies. The implementation of such algorithms in security tools such as the IBM Enterprise Security Policy Evaluator offers the potential for comprehensive and consistent policy evaluation.
- **Suggestions for Fixing Defects:** Automated analysis can suggest alternative policies that address identified vulnerabilities, enabling rapid and informed security policy improvements without the need for extensive manual intervention.

Challenges:

- **Complexity of Policy Validation:** The complexity of security policy validation, especially in large and complex systems, can be a significant challenge. Automated tools must account for different scenarios and role requirements, potentially increasing the complexity of the analysis algorithm.
- **Boundaries of Analysis Techniques:** Although static analysis can uncover many policy problems, it may have limitations in certain contexts. For example, the analysis may rely on specific techniques, such as Rapid Type Analysis (RTA), which may have limitations on the types of applications or security flaws that it can effectively evaluate.
- **Addressing Runtime Authorization Failures:** Ensuring automatic stability and preventing runtime authorization failures remains a challenge. The system must be robust enough to handle dynamic scenarios where user roles and permissions may change on the fly.
- **Managing Intra-Component Calls:** The issue of subversive policies that exploit unchecked intra-component calls requires careful analysis to ensure that execution points within components conform to the overall security policy and do not lead to security breaches.

Bibliography

[10] "ISO/IEC/IEEE International Standard - Systems and software engineering – Vocabulary". In: *ISO/IEC/IEEE 24765:2010(E)* (2010), pp. 1–418. DOI: 10.1109/IEEESTD.2010.5733835 (cit. on pp. 1, 9).

[13] "Automatic data protection in a computer system". 2013 (cit. on p. 2).

[14] *Automatic reconstruction and analysis of security policies from deployed security components.* 2014 (cit. on pp. 1, 2).

[18] *Stakeholder Concern-Driven Requirements Analytics.* 2018. DOI: 10.1145/3178315.3178324 (cit. on p. 22).

[19] *Motivational and Goal-Oriented Viewpoint for Architectural Modeling of Software Intensive Systems.* 2019. DOI: 10.1007/978-3-030-30275-7_9 (cit. on p. 23).

[21] *A comprehensive approach to identifying key stakeholders in complicated software ecosystems.* 2021. DOI: 10.1109/RE51729.2021.00074 (cit. on p. 22).

[22a] "ISO/IEC 27002:2022 Information security, cybersecurity and privacy protection". In: *ISO/IEC 27002:2022* (2022) (cit. on p. 6).

[22b] "ISO/IEC/IEEE 42010 Software, systems and enterprise — Architecture description". In: *ISO/IEC/IEEE 42010:2022(E)* (2022) (cit. on pp. 24, 25).

[AA19] A. Al-Haj and B. Aziz. "Enforcing Multilevel Security Policies in Database-Defined Networks using Row-Level Security". In: *2019 International Conference on Networked Systems (NetSys).* Mar. 2019, pp. 1–6. DOI: 10.1109/NetSys.2019.8854491 (cit. on pp. 95, 131).

[AE08a] R. Abassi and S. G. El Fatmi. "A Model for Specification and Validation of Security Policies in Communication Networks: The Firewall Case". In: *2008 Third International Conference on Availability, Reliability and Security.* Mar. 2008, pp. 467–472. DOI: 10.1109/ARES.2008.124 (cit. on pp. 93, 129).

[AE08b] R. Abassi and S. G. El Fatmi. "An Automated Validation Method for Security Policies: The Firewall Case". In: *2008 The Fourth International Conference on Information Assurance and Security.* Sept. 2008, pp. 291–294. DOI: 10.1109/IAS.2008.52 (cit. on pp. 1, 7, 88, 121).

[AF11] R. Abassi and S. G. E. Fatmi. "Using security policies in a network securing process". In: *2011 18th International Conference on Telecommunications.* May 2011, pp. 416–421. DOI: 10.1109/CTS.2011.5898961 (cit. on pp. 93, 129).

[AG08] R. Abassi and S. Guemara El Fatmi. "Towards an automated firewall security policies validation process". In: *2008 Third International Conference on Risks and Security of Internet and Systems*. Oct. 2008, pp. 267–272. DOI: 10.1109/CRISIS.2008.4757489 (cit. on pp. 102, 153).

[AGL03] G. Angelis, S. Gritzalis, and C. Lambrinoudakis. "Security Policy Configuration Issues in Grid Computing Environments". In: *Computer Safety, Reliability, and Security*. Ed. by S. Anderson, M. Felici, and B. Littlewood. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 331–341. ISBN: 978-3-540-39878-3 (cit. on pp. 106, 164, 165).

[ALP03a] M. Archer, E. Leonard, and M. Pradella. "Analyzing security-enhanced Linux policy specifications". In: *Proceedings POLICY 2003. IEEE 4th International Workshop on Policies for Distributed Systems and Networks*. June 2003, pp. 158–169. DOI: 10.1109/POLICY.2003.1206969 (cit. on p. 96).

[ALP03b] M. Archer, E. Leonard, and M. Pradella. "Modeling security-enhanced Linux policy specifications for analysis". In: *Proceedings DARPA Information Survivability Conference and Exposition*. Vol. 2. Apr. 2003, 164–169 vol.2. DOI: 10.1109/DISCEX.2003.1194959 (cit. on p. 94).

[Alv+15] J. Alves-Foss et al. "Evaluating the Use of Security Tags in Security Policy Enforcement Mechanisms". In: *2015 48th Hawaii International Conference on System Sciences*. Jan. 2015, pp. 5201–5210. DOI: 10.1109/HICSS.2015.614 (cit. on pp. 95, 132).

[Bas+15] C. Basile et al. "A novel approach for integrating security policy enforcement with dynamic network virtualization". In: *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. Apr. 2015, pp. 1–5. DOI: 10.1109/NETSOFT.2015.7116152 (cit. on pp. 10, 96, 133, 135).

[BBR15] A. Basak, S. Bhunia, and S. Ray. "A Flexible Architecture for Systematic Implementation of SoC Security Policies". In: *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. ICCAD '15. Austin, TX, USA: IEEE Press, 2015, pp. 536–543. ISBN: 9781467383899 (cit. on pp. 16, 17, 83, 110).

[Ber+20] D. Berardi et al. "TechNETium: Atomic Predicates and Model Driven Development to Verify Security Network Policies". In: *2020 IEEE 17th Annual Consumer Communications Networking Conference (CCNC)*. Jan. 2020, pp. 1–6. DOI: 10.1109/CCNC46108.2020.9045145 (cit. on pp. 92, 127).

[BSV22] S. Bussa, R. Sisto, and F. Valenza. "Security Automation using Traffic Flow Modeling". In: *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*. 2022, pp. 486–491. DOI: 10.1109/NetSoft54395.2022.9844025 (cit. on p. 7).

[CC04] H. Chen and S. Chong. "Owned policies for information security". In: *Proceedings. 17th IEEE Computer Security Foundations Workshop, 2004.* June 2004, pp. 126–138. DOI: 10.1109/CSFW.2004.1310737 (cit. on pp. 101, 152).

[CCH03] S. Chang, Q. Chen, and M. Hsu. "Managing security policy in a large distributed Web services environment". In: *Proceedings 27th Annual International Computer Software and Applications Conference. COMPAC 2003.* Nov. 2003, pp. 610–621. DOI: 10.1109/CMPSAC.2003.1245404 (cit. on pp. 91, 125).

[Chr+14] M. Chraibi et al. "Policy Based Security Middleware as a Service". In: *2014 International Conference on Future Internet of Things and Cloud.* Aug. 2014, pp. 322–327. DOI: 10.1109/FiCloud.2014.57 (cit. on pp. 94, 131).

[CS13] J. Coertze and R. von Solms. "A software gateway to affordable and effective information security governance in SMMEs". In: *2013 Information Security for South Africa.* IEEE. 2013, pp. 1–8 (cit. on p. 6).

[Dur+17] L. Durante et al. "A model for the analysis of security policies in service function chains". In: *2017 IEEE Conference on Network Softwarization (NetSoft).* July 2017, pp. 1–6. DOI: 10.1109/NETSOFT.2017.8004230 (cit. on pp. 91, 126).

[Eye+08] D. M. Eyers et al. "Compile-Time Enforcement of Dynamic Security Policies". In: *2008 IEEE Workshop on Policies for Distributed Systems and Networks.* June 2008, pp. 119–126. DOI: 10.1109/POLICY.2008.24 (cit. on pp. 97, 136).

[Ger18] C. Gerking. "Specification of Information Flow Security Policies in Model-Based Systems Engineering". In: *Software Technologies: Applications and Foundations.* Ed. by M. Mazzara, I. Ober, and G. Salaun. Cham: Springer International Publishing, 2018, pp. 617–632. ISBN: 978-3-030-04771-9 (cit. on pp. 103, 158).

[Gup+12] A. Gupta et al. "Intuitive Security Policy Configuration in Mobile Devices Using Context Profiling". In: *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing.* Sept. 2012, pp. 471–480. DOI: 10.1109/SocialCom-PASSAT.2012.60 (cit. on pp. 99, 144, 145).

[H19] R. H. "Security Policy Modelling in the Mobile Agent System". In: *International Journal of Computer Network and Information Security* 11.10 (2019), pp. 26–36 (cit. on pp. 16, 17, 103, 156, 157).

[HAC19] S. Hsaini, S. Azzouzi, and M. E. H. Charaf. "FSM Modeling of Testing Security Policies for MapReduce Frameworks". In: *2019 6th International Conference on Control, Decision and Information Technologies (CoDIT).* Apr. 2019, pp. 1480–1485. DOI: 10.1109/CoDIT.2019.8820685 (cit. on pp. 15, 87, 118).

[Hag+18] M. Hagan et al. "Enforcing Policy-Based Security Models for Embedded SoCs within the Internet of Things". In: *2018 IEEE Conference on Dependable and Secure Computing (DSC)*. Dec. 2018, pp. 1–8. DOI: 10.1109/DESEC.2018.8625140 (cit. on pp. 1, 16, 17, 85, 114, 115).

[Har+05] P. Hartel et al. "Modelling Mobility Aspects of Security Policies". In: *Construction and Analysis of Safe, Secure, and Interoperable Smart Devices*. Ed. by G. Barthe et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 172–191. ISBN: 978-3-540-30569-9 (cit. on pp. 105, 163).

[HBM07] H. HAMDI, A. Bouhoula, and M. Mosbah. "A Software Architecture for Automatic Security Policy Enforcement in Distributed Systems". In: *The International Conference on Emerging Security Information, Systems, and Technologies (SECUREWARE 2007)*. Oct. 2007, pp. 187–192. DOI: 10.1109/SECUREWARE.2007.4385332 (cit. on pp. 1, 2, 7, 10, 98, 139).

[Her03] P. Herrmann. "Formal Security Policy Verification of Distributed Component-Structured Software". In: *Formal Techniques for Networked and Distributed Systems - FORTE 2003*. Ed. by H. Knig, M. Heiner, and A. Wolisz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 257–272. ISBN: 978-3-540-39979-7 (cit. on pp. 106, 166).

[HK01] P. Herrmann and H. Krumm. "Trust-adapted enforcement of security policies in distributed component-structured applications". In: *Proceedings. Sixth IEEE Symposium on Computers and Communications*. July 2001, pp. 2–8. DOI: 10.1109/ISCC.2001.935347 (cit. on pp. 88, 119).

[HPF14] J.-M. Horcas, M. Pinto, and L. Fuentes. "Closing the Gap between the Specification and Enforcement of Security Policies". In: *Trust, Privacy, and Security in Digital Business*. Ed. by C. Eckert, S. K. Katsikas, and G. Pernul. Cham: Springer International Publishing, 2014, pp. 106–118. ISBN: 978-3-319-09770-1 (cit. on pp. 7, 9, 105, 161).

[HS23] K. Hammar and R. Stadler. "Digital Twins for Security Automation". In: *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*. 2023, pp. 1–6. DOI: 10.1109/NOMS56928.2023.10154288 (cit. on p. 7).

[Hwa+12] J. Hwang et al. "Selection of Regression System Tests for Security Policy Evolution". In: *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. ASE '12. Essen, Germany: Association for Computing Machinery, 2012, pp. 266–269. ISBN: 9781450312042. DOI: 10.1145/2351676.2351719. URL: https://doi.org/10.1145/2351676.2351719 (cit. on pp. 81, 110).

[IDC07] I. Ion, B. Dragovic, and B. Crispo. "Extending the Java Virtual Machine to Enforce Fine-Grained Security Policies in Mobile Devices". In: *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*. Dec. 2007, pp. 233–242. DOI: 10.1109/ACSAC.2007.36 (cit. on pp. 100, 147).

[IKC11] W. Itani, A. Kayssi, and A. Chehab. "Policy-based security channels for protecting network communication in mobile cloud computing". In: *Proceedings of the International Conference on Security and Cryptography*. July 2011, pp. 450–456 (cit. on pp. 89, 122).

[Jau10] M. Jaume. "Security Rules versus Security Properties". In: *INFORMATION SYSTEMS SECURITY*. Ed. by S. Jha and A. Mathuria. Vol. 6503. Lecture Notes in Computer Science. 6th International Conference on Information Systems Security, Gandhinagar, INDIA, DEC 17-19, 2010. 2010, pp. 231–245. ISBN: 978-3-642-17713-2 (cit. on p. 9).

[Kaf+17] Ö. Kafali et al. "How Good Is a Security Policy against Real Breaches? A HIPAA Case Study". In: *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. May 2017, pp. 530–540. DOI: 10.1109/ICSE.2017.55 (cit. on pp. 15, 83).

[Kee+07] S. Keele et al. *Guidelines for performing systematic literature reviews in software engineering*. 2007 (cit. on p. 11).

[KG18] R. Khelf and N. Ghoualmi-Zine. "IPsec/Firewall Security Policy Analysis: A Survey". In: *2018 International Conference on Signal, Image, Vision and their Applications (SIVA)*. Nov. 2018, pp. 1–7. DOI: 10.1109/SIVA.2018.8660973 (cit. on p. 101).

[Klä+10] M. Kläs et al. "How to evaluate meta-models for software quality". In: *Proceedings of the 20th International Workshop on Software Measurement (IWSM2010)*. 2010 (cit. on p. 74).

[LD08] Y. Liang and W. Deng. "Verify Consistency between Security Policy and Firewall Policy with Answer Set Programming". In: *2008 International Conference on Computer Science and Software Engineering*. Vol. 1. Dec. 2008, pp. 196–200. DOI: 10.1109/CSSE.2008.1089 (cit. on pp. 90, 123).

[LK17] W. Lee and N. Kim. "Security Policy Scheme for an Efficient Security Architecture in Software-Defined Networking". In: *Information* 8.2 (2017). ISSN: 2078-2489. DOI: 10.3390/info8020065. URL: https://www.mdpi.com/2078-2489/8/2/65 (cit. on pp. 104, 160).

[LL11] D. Lomsak and J. Ligatti. "PoliSeer: A Tool for Managing Complex Security Policies". In: *Journal of Information Processing* 19 (2011), pp. 292–306. DOI: 10.2197/ipsjjip.19.292 (cit. on pp. 105, 163).

[LR16] A. Lara and B. Ramamurthy. "OpenSec: Policy-Based Security Using Software-Defined Networking". In: *IEEE Transactions on Network and Service Management* 13.1 (Mar. 2016), pp. 30–42. ISSN: 1932-4537. DOI: 10.1109/TNSM.2016.2517407 (cit. on pp. 10, 15, 17, 97, 137).

[Mig+10] M. Migliavacca et al. "Distributed Middleware Enforcement of Event Flow Security Policy". In: *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware*. Middleware '10. Bangalore, India: Springer-Verlag, 2010, pp. 334–354. ISBN: 9783642169540 (cit. on pp. 15–17, 81, 109).

[MP03] P. McDaniel and A. Prakash. "A flexible architecture for security policy enforcement". In: *Proceedings DARPA Information Survivability Conference and Exposition*. Vol. 2. Apr. 2003, 234–239 vol.2. DOI: 10.1109/DISCEX.2003.1194971 (cit. on p. 92).

[MRF21] S. Mohammad, M. M. M. Rahman, and F. Farahmandi. "Required Policies and Properties of the Security Engine of an SoC". In: *2021 IEEE International Symposium on Smart Electronic Systems (iSES)*. Dec. 2021, pp. 414–420. DOI: 10.1109/iSES52644.2021.00100 (cit. on pp. 9, 17, 86, 116).

[NC16] J. Navarro-Machuca and L.-C. Chen. "Embedding Model-Based Security Policies in Software Development". In: *2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS)*. Apr. 2016, pp. 116–122. DOI: 10.1109/BigDataSecurity-HPSC-IDS.2016.46 (cit. on pp. 1, 2, 7, 9, 89).

[NIC08] E. W. NICK ROZANSKI. *SOFTWARE SYSTEMS ARCHITECTURE*. 2008 (cit. on pp. 19, 25).

[PCH16] S. Pisharody, A. Chowdhary, and D. Huang. "Security policy checking in distributed SDN based clouds". In: *2016 IEEE Conference on Communications and Network Security (CNS)*. Oct. 2016, pp. 19–27. DOI: 10.1109/CNS.2016.7860466 (cit. on pp. 100, 149).

[Per19] J. Persson. *Stakeholder a Developer Communication Powered by Product Owners*. 2019 (cit. on p. 23).

[Pis+07a] M. Pistoia et al. "When Role Models Have Flaws: Static Validation of Enterprise Security Policies". In: *29th International Conference on Software Engineering (ICSE'07)*. May 2007, pp. 478–488. DOI: 10.1109/ICSE.2007.98 (cit. on pp. 16, 84, 111).

[Pis+07b] M. Pistoia et al. "When role models have flaws: Static validation of enterprise security policies". In: *ICSE 2007: 29TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING,PROCEEDINGS*. International Conference on Software Engineering. 29th International Conference on Software Engineering (ICSE 2007),Minneapolis, MN, MAY 20-26, 2007. IEEE Comp Soc, TCSE; ACM SIGSOFT. 2007, pp. 478+. ISBN: 978-0-7695-2828-1 (cit. on pp. 9, 107, 166, 167).

[Pis+19] S. Pisharody et al. "Brew: A Security Policy Analysis Framework for Distributed SDN-Based Cloud Environments". In: *IEEE Transactions on Dependable and Secure Computing* 16.6 (Nov. 2019), pp. 1011–1025. ISSN: 1941-0018. DOI: 10.1109/TDSC.2017.2726066 (cit. on pp. 17, 99, 143).

[PS08] P. H. Phung and D. Sands. "Security Policy Enforcement in the OSGi Framework Using Aspect-Oriented Programming". In: *2008 32nd Annual IEEE International Computer Software and Applications Conference*. July 2008, pp. 1076–1082. DOI: 10.1109/COMPSAC.2008.149 (cit. on pp. 101, 151).

[PTN09] T.-H. Pham, N.-T. Truong, and V.-H. Nguyen. "Analyzing RBAC Security Policy of Implementation Using AST". In: *2009 International Conference on Knowledge and Systems Engineering*. Oct. 2009, pp. 215–219. DOI: 10.1109/KSE.2009.23 (cit. on pp. 1, 92, 128).

[RKG22] E. Rostami, F. Karlsson, and S. Gao. "Policy Components - A Conceptual Model for Tailoring Information Security Policies". In: *Human Aspects of Information Security and Assurance*. Ed. by N. Clarke and S. Furnell. Cham: Springer International Publishing, 2022, pp. 265–274. ISBN: 978-3-031-12172-2 (cit. on pp. 1, 2, 5, 6, 71, 103, 155).

[Rou+23] Q. Rouland et al. "Formalizing the Relationship between Security Policies and Objectives in Software Architectures". In: *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*. Mar. 2023, pp. 151–158. DOI: 10.1109/ICSA-C57050.2023.00042 (cit. on pp. 1, 2, 7, 86, 116).

[RS17] S. Ranise and H. Siswantoro. "Automated Legal Compliance Checking by Security Policy Analysis". In: *Computer Safety, Reliability, and Security*. Ed. by S. Tonetta, E. Schoitsch, and F. Bitsch. Cham: Springer International Publishing, 2017, pp. 361–372. ISBN: 978-3-319-66284-8 (cit. on pp. 7, 104, 159).

[SG14] M. Salnitri and P. Giorgini. "Modeling and verification of ATM security policies with SecBPMN". In: *2014 International Conference on High Performance Computing Simulation (HPCS)*. July 2014, pp. 588–591. DOI: 10.1109/HPCSim.2014.6903740 (cit. on pp. 84, 112).

[Sin22] B. Sinkovec. *Towards a software engineering view of security for microservice-based applications*. 2022 (cit. on pp. 5, 9, 11, 21, 45, 46).

[Soo+19] K. Sood et al. "Analysis of Policy-Based Security Management System in Software-Defined Networks". In: *IEEE Communications Letters* 23.4 (Apr. 2019), pp. 612–615. ISSN: 1558-2558. DOI: 10.1109/LCOMM.2019.2898864 (cit. on pp. 17, 90, 124).

[SY13] Z. Sainan and H. Yu. "Research and application of XACML-based fine-grained security policy for distributed system". In: *Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC)*. Dec. 2013, pp. 1848–1851. DOI: 10.1109/MEC.2013.6885354 (cit. on pp. 15, 16, 85, 113).

[Sya10] I. Syamsuddin. "The use of AHP in security policy decision making: an open office calc application". In: *Journal of Software* 5.10 (2010) (cit. on p. 6).

[Var+19] V. Varadharajan et al. "A Policy-Based Security Architecture for Software-Defined Networks". In: *IEEE Transactions on Information Forensics and Security* 14.4 (Apr. 2019), pp. 897–912. ISSN: 1556-6021. DOI: 10.1109/TIFS.2018.2868220 (cit. on pp. 98, 141).

[Wro+17] K. Wrona et al. "SDN testbed for validation of cross-layer data-centric security policies". In: *2017 International Conference on Military Communications and Information Systems (ICMCIS)*. May 2017, pp. 1–6. DOI: 10.1109/ICMCIS.2017.7956483 (cit. on pp. 9, 17, 88).

[XM03] W. Xie and H. Ma. "A policy-based security model for Web system". In: *International Conference on Communication Technology Proceedings, 2003. ICCT 2003*. Vol. 1. Apr. 2003, 187–191 vol.1. DOI: 10.1109/ICCT.2003.1209065 (cit. on pp. 15, 87, 117).

[YZG17] S. T. Yakasai, F.-C. Zheng, and C. G. Guy. "Towards policy unification for enterprise network security". In: *2017 IEEE Conference on Network Softwarization (NetSoft)*. July 2017, pp. 1–5. DOI: 10.1109/NETSOFT.2017.8004205 (cit. on pp. 10, 17, 94, 130).

[ZHM10] Q. Zhang, Y. He, and C. Meng. "Semantic Remote Attestation for Security Policy". In: *2010 International Conference on Information Science and Applications*. Apr. 2010, pp. 1–8. DOI: 10.1109/ICISA.2010.5480265 (cit. on pp. 102, 154).

Glossary

AM Automation Model

BPMN Business Process Model and Notation

CP Construction Process

SDLC Software Development Life Cycle

SLR Systematic Literature Review

SPACM Security Policy Automation Concept Model

SPAM Security Policy Automation Model

SPCM Security Policy Concept Model

SPM Security Policy Model

SPMM Security Policy Meta Model

UML Unified Modeling Language