RBAC Policy Engineering with Patterns

Taufiq Rochaeli* and Claudia Eckert

Departement of Computer Science
Darmstadt University of Technology
{rochaeli,eckert}@sec.informatik.tu-darmstadt.de

Abstract. We present a RBAC policy engineering approach that supports administrators to specify RBAC policies with the help of experts' knowledge, which is documented using the pattern paradigm. These patterns are formalised in Web Ontology Language (OWL) that enables machine interpretation of experts' knowledge and reasoning about the RBAC policy. Thus, administrators could specify RBAC policies by choosing the patterns matching their scenario and asserting instances without knowing the complex RBAC policy specification.

1 Introduction

Role-Based Access Control (RBAC) [1] offers a better manageability than traditional Mandatory Access Control (MAC) or Discretionary Access Control (DAC) to fulfil organisational security policies. However, in an organisation with complex scenarios, the specification of access control policies using role based model may also be overwhelming for security administrators. They need domain as well as security experts' knowledge to accomplish this task. This knowledge provides security administrators with well-proven empirical solutions to access control policies specification within a certain domain (i.e. hospital, government, etc.), which is strongly affected by social and/or economical factors. Furthermore, increasing complexity of the scenario also poses another inconsistency problem, which is caused by the manual specification of RBAC policies. For this reason, a tool support to generate RBAC policies specification is required.

We developed a role engineering approach, which follows the scenario-oriented requirements engineering approach [2] that assists security administrators to specify the RBAC policies with the help of experts' knowledge. As argued above, the experts' knowledge on specifying RBAC policies with consideration on several factors in a certain domain will greatly helps security administrators to specify RBAC policies.

To document the experts' knowledge, we follow the structure proposed in pattern paradigm originated from the architecture field [3]. This paradigm has been widely used in several fields in computer science [4, 5]. Despite of the improvement in their engineering processes, pattern paradigm still has a major

* partially supported by the German Ministry of Education and Research (BMBF) under SicAri Project.

drawback: novice users find it difficult to interpret and to apply the patterns. By encoding experts' knowledge in specifying RBAC policies in semantic language, i.e. OWL, we explore the possibilities of automatic generation of RBAC policies by using reasoning services provided by description logic-based knowledge representation system. With this framework, security administrators only need to interpret their scenario, search for patterns that match the scenario and assert instances in the scenario into the knowledge representation system. Hence, the complexity of RBAC policies specification could be concealed by the abstraction of more understandable concepts of the scenario.

This paper is organised as follows. Section 2 begins with the background of this work and subsequently presents the definition of RBAC policy patterns in description logic notation[6]. Section 3 outlines the generation process of RBAC policies by using the reasoning services of knowledge representation system. Section 4 discusses some related works in pattern paradigm and role engineering. Finally, section 5 presents the conclusion.

2 Definition of RBAC Policy Pattern

This section briefly describes the background of pattern paradigm and continues with the motivation that drives the adaptation of pattern paradigm in RBAC policies specification process. It explains the similarities between construction design and RBAC policies specification and continues with the definition of RBAC policies pattern.

To design a building, architects should consider many non-technical human-related factors, for example, social and/or psychology factors. This is necessary, due to the fact that people live and interact in the building and have their own needs. Therefore, architects should design constructions, which fulfil the needs emerging from these factors.

Christopher Alexander proposed *pattern* method in his work [3], which structurally captures experts' knowledge. Each pattern has three general parts, a **context**, a **problem** and a **solution**. The context describes the environment, in which the pattern shall apply. It covers both temporal and spatial aspects of the environment that represent a scenario. In this context, there exist forces[1] that need to be resolved. These forces characterise the problem in the context. Finally, the solution proposes a configuration or a design which should resolves the existing forces in the context. Collected patterns do not exist independently; they have one or more relationships with other patterns. These relationships reflect the conflicts, compatibilities and dependencies between patterns applied within a context.

The pattern approach in software engineering [4] and security engineering [5], which captures the solutions considering different non-technical factors, motivates this work to adapt the pattern approach in RBAC policy specification. Thus, RBAC policy pattern shall capture this expertise knowledge, which has

[1] In our work, we interpret forces as requirements and problems.

the solution to specify access control policies in a certain situation by considering various non-technical factors, even the subtle one.

A pattern consists of a context with problem and a solution: $\mathsf{Pattern} \equiv \exists \mathit{hasCtx}.\mathsf{Context} \sqcap \exists \mathit{hasSln}.\mathsf{Solution}$. In the next two subsections we define the main parts of RBAC policy pattern, which are described using description logic notation.

2.1 Context and Problem

The context of RBAC policy pattern represents a novel description of business process scenario. It has SubjectInTaskClass and ObjectClass as main concepts, which are interpreted as a class of subject performing a certain task, and as a class of resources, respectively. Optionally, the context may also have concepts and concepts relationships of the Event-controlled Process Chain [7] such as Event and *controlFlow*, which links Event with SubjectInTaskClass.

The critical aspect of context of business process is the information that flows from subjects to objects and vice versa. Information flow is needed in order to perform the business process. On the other hand, a possible unintended information flow could also pose a security threat to the business process. We identify two main classes of problem, which arise in this context. They are *workflow requirements* and *security threats*. These problems are denoted by concept relationships between SubjectInTaskClass and ObjectClass. A workflow requirement relationship means that a subject needs an access to objects in order to perform the business process. The workflow relationship is further classified into *hasWflowReqInput* and *hasWflowReqOutput*, which represent the input and output information flow needed in business process between the subjects in a task and the objects. A security threat relationship means that a subject could perform an attack to objects, which can cause any harm to the business process. The security attack relationship is also classified into two classes, *hasSecThreatInput* and *hasSecThreatOutput*, which represent the input and output information flow between the subjects in a task and the objects posing security threats.

Fig. 1 shows an excerpt of transaction pattern context adapted from the reference model of an industrial business process in [8] with its problem. In this context, a malicious worker in WarehouseManagement task could issue a fictive transaction, so that he can steal some goods in the warehouse. This threat is represented by *createFictiveTransactionIn*. Note that, we do not define problem as separate concept, because the problem is already represented by concept relationships.

It is also possible to define a context having conflicting forces, i.e., a task class requires a write access to a database, but it also poses a security threat when it write to the database. To prevent this definition, both axioms

$\neg(\exists \mathit{hasTask}.\exists \mathit{hasSecThreatInput}.\mathsf{ObjectClass}) \sqsupseteq \exists \mathit{hasTask}.\exists \mathit{hasWflowReqInput}.\mathsf{ObjectClass}$

and

$\neg(\exists \mathit{hasTask}.\exists \mathit{hasSecThreatOutput}.\mathsf{ObjectClass}) \sqsupseteq \exists \mathit{hasTask}.\exists \mathit{hasWflowReqOutput}.\mathsf{ObjectClass}$

should be defined.

$$
\begin{aligned}
\mathsf{TransactionCtx} &\equiv \exists \mathit{hasTask}.\mathsf{TransactionMonitoring} \sqcap \exists \mathit{hasTask}.\mathsf{WarehouseManagement} \sqcap \ \ldots \\
&\quad\ \exists \mathit{hasObject}.\mathsf{Transaction} \sqcap \exists \mathit{hasObject}.\mathsf{DunningLetter} \sqcap \\
&\quad\ \exists \mathit{hasObject}.\mathsf{DeliveryNote} \sqcap \exists \mathit{hasObject}.\mathsf{Inventory} \sqcap \ \ldots \\
\mathit{createFictiveTransactionIn} &\sqsubseteq \mathit{hasSecThreatOutput} \\
\mathsf{WarehouseManagement} &\equiv \exists \mathit{createFictiveTransactionIn}.\mathsf{Transaction} \sqcap \exists \mathit{hasWflowReqInput}.\mathsf{Transaction} \sqcap \ldots \\
\mathsf{TransactionMonitoring} &\equiv \exists \mathit{hasWflowReqInput}.\mathsf{Transaction} \sqcap \exists \mathit{hasWflowReqInput}.\mathsf{DunningLetter} \sqcap \ldots
\end{aligned}
$$

Fig. 1. Context and problem of *Transaction* pattern

2.2 Solution

The solution part of a RBAC policy pattern specifies authorisation policies of a context that fulfil the requirements within this context. Currently, the solution only proposes the core RBAC and hierarchical RBAC specification of the RBAC INCITS standard [1].

A solution defines permission and role concepts, which have different interpretation than the concepts introduced in RBAC model. We interpret *permission* as a class of subjects which have a certain permission. For example, $\mathsf{P1} \equiv \exists \mathit{read}.\mathsf{DatabaseX}$ is a class of subjects which are permitted to read database X. A *role* is interpreted as a class of subjects which have required permissions to perform a task. A role concept is constructed by intersection of different permissions. With our definition, $\mathsf{R2} \sqsubseteq \mathsf{R1}$ is interpreted by DL reasoning service as: (1) all permissions of R2 are included in R1, (2) all users in R2 are also members of R1. Thus, role hierarchy relationship $\mathsf{R2} \preceq \mathsf{R1}$ could be represented by inclusion $\mathsf{R2} \sqsubseteq \mathsf{R1}$. In case of redundant definition of permissions among roles, role hierarchy could be automatically detected and built by using classification service provided by reasoning engine.

From the context, the administrator already knows the membership of subjects in task classes. We extend the interpretation of SubjectInTaskClass to a class of subjects performing a certain task and having the necessary role(s). Therefore, the solution defines SubjectInTaskClass as intersection of roles. This kind of definition implicitely defines $\mathsf{SubjectInTaskClass} \sqsubseteq \mathsf{Role}$, which ensures that every subjects having task membership are always assigned to roles.

Fig. 2 shows an excerpt of transaction pattern solution.

$$
\begin{aligned}
\mathsf{TransactionSln} &\equiv \exists \mathit{defines}.\mathsf{ManufacturingRole} \sqcap \exists \mathit{defines}.\mathsf{ShippingDeptRole} \sqcap \ldots \\
\mathsf{ManufacturingRole} &\equiv \exists \mathit{get}.\mathsf{DunningLetter} \sqcap \exists \mathit{put}.\mathsf{DunningLetter} \sqcap \exists \mathit{get}.\mathsf{Transaction} \\
\mathsf{ShippingDeptRole} &\equiv \exists \mathit{get}.\mathsf{Transaction} \sqcap \exists \mathit{put}.\mathsf{DeliveryNote} \sqcap \exists \mathit{put}.\mathsf{Inventory} \\
\mathsf{TransactionMonitoring} &\equiv \mathsf{ManufacturingRole} \\
\mathsf{WarehouseManagement} &\equiv \mathsf{ShippingDeptRole}
\end{aligned}
$$

Fig. 2. Solution of *Transaction* pattern

2.3 Relationship between RBAC Policy Patterns

The relationships between patterns distinguish pattern from template. They represent compatibility between patterns and guide the pattern application process. The compatibility between patterns is represented by *refinement* and *dependency* relationships. The *conflict* relationship should guide the pattern user to avoid simultaneous use of these patterns, which have conflicting requirements between pattern contexts. In this paper, we only focus on *conflict* relationship. A pattern can conflict with another pattern, if and only if, the contexts of these patterns have any task class, which has a security threat relationship in one context, and also has a workflow requirement relationship in another context to the same object class. This relationship is formally defined in fig. 3.

$$
\begin{aligned}
confl \equiv\ & hasCtx \circ \\
& ((hasTask \circ hasSecThreatInput \circ hasObject^{-} \sqcap hasObject \circ hasWorkflowReqInput^{-} \circ hasTask^{-}) \sqcup \\
& (hasTask \circ hasSecThreatOutput \circ hasObject^{-} \sqcap hasObject \circ hasWorkflowReqOutput^{-} \circ hasTask^{-})) \circ \\
& hasCtx^{-}
\end{aligned}
$$

Fig. 3. Conflict relationship

3 Generating RBAC policies

In order to generate RBAC policies, security administrator should first interpret his scenario. Next, he chooses patterns starting from the most general one to the detailed one. The selection criterion is that the context of patterns should (partially) match the concrete scenario and (partially) fulfil the requirements of scenario. For each selected pattern, concept instances of pattern, concept and solution and concept relationships should be asserted.

After all requirements of scenario have been met by the patterns, he asserts the instances of scenario matching the context concepts into the knowledge representation system. Subject and role assignments are be defined by retrieving instances of roles. Role and permission assignments are defined by retrieving concept descendants of role.

4 Related Works

Previous work on formalisation of patterns [9, 5] propose formalisation of patterns based on first-order predicate logic and frame-logic, respectively. The first work only defines the formalisation of pattern *solution* without its related *context* and *problem*. It also lacks of definition of pattern relationship. The latter work only defines the formalisation of pattern *structure* and its relationships. In comparison of previous works to our case, we encode the context, problem and the solution in semantic language based on description logic. Therefore, machine interpretation of context, problem and solution is possible. However, in our current

work, searching and selecting suitable patterns is still done by involving human intelligence, which interprets the context of pattern. Since the description logic notation of context is still hard to understand, each pattern also provides the description of its context as plain text. In the role engineering area, Neumann et. al. present in [10] a method to derive role from scenario. Our approach differs in a way that the expertise knowledge in a certain domain is documented using pattern paradigm instead of catalog.

5 Conclusion

In our approach, the involvement of the policy designer in the RBAC policies specification task is reduced only to scenario identification, patterns selection and instances assertion. Thus, the possibility of human error in RBAC policy specification that could lead into inconsistent and redundant specification could be avoided.

Acknowledgements

The authors thank anonymous reviewers for helpful comments.

References

1. American National Standards Institute: ANSI Standard 359-2004: Role Based Access Control (2004)
2. Alistair G. Sutcliffe, Neil A.M. Maiden, Shailey Minocha, Darrel Manuel: Supporting scenario-based requirements engineering. IEEE Transactions on Software Engineering **24** (1998) 1072 – 1088
3. Alexander, C.: The Timeless Way of Building. Oxford University Press (1979)
4. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns. Addison Wesley (1995)
5. Schumacher, M.: Security Engineering with Patterns - Origins, Theoretical Model, and New Applications. Volume 2754 of LNCS. Springer (2003)
6. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P., eds.: The Description Logic Handbook. Cambridge University Press (2004)
7. (G. Keller, M. Nüttgens, A.-W. Scheer)
8. A. -W. Scheer: Wirtschaftsinformatik: Referenzmodelle für industrielle Geschäftsprozesse(in German). Springer (1998)
9. Alencar, P., Cowan, D., Dong, J., Lucena, C.: A pattern-based approach to structural design composition. In: Twenty-Third Annual International Computer Software and Applications Conference, IEEE Press (1999)
10. Neumann, G., Strembeck, M.: A scenario-driven role engineering process for functional rbac roles. In: SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies, New York, NY, USA, ACM Press (2002) 33–42