Supporting Agile Development of Authorization Rules for SME Applications

Steffen Bartsch, Karsten Sohr, and Carsten Bormann

Technologie-Zentrum Informatik TZI,
Universität Bremen, Bibliothekstr. 1, 28359 Bremen, Germany
`{sbartsch,sohr,cabo}@tzi.org`

Abstract. Custom SME applications for collaboration and workflow have become affordable when implemented as Web applications employing Agile methodologies. Security engineering is still difficult with Agile development, though: heavy-weight processes put the improvements of Agile development at risk. We propose Agile security engineering and increased end-user involvement to improve Agile development with respect to authorization policy development. To support the authorization policy development, we introduce a simple and readable authorization rules language implemented in a Ruby on Rails authorization plugin that is employed in a real-world SME collaboration and workflow application. Also, we report on early findings of the language's use in authorization policy development with domain experts.

Key words: Authorization Policy, Agile Security Engineering, End-User Development, DSL, SME Applications

1 Introduction

When Small and Medium Enterprises (SME) deploy collaboration and workflow applications, the applications need to measure up to the established workflows in terms of efficiency and flexibility. SMEs are often incapable of investing the required resources into tailoring commercial off-the-shelf software to match the established workflows. This is further backed by the observation that it is often the unique selling point of SMEs to implement unconventional processes when compared to competing larger companies. With the advent of recent technological developments in the Web sector, small and focussed custom applications have become affordable for implementing SMEs' specific needs in collaboration and workflow management in *SME applications*.

One aspect of the development of custom SME applications is implementing authorization. A large amount of research has been invested into the authorization realm resulting e.g. in *Role-based Access Control* (RBAC, [9, 15, 2]). Specific solutions have been proposed for collaboration and workflow [4, 14, 18, 16] as well as high flexibility [19]. Still, with respect to SME applications, the established approaches are not easily implemented in practice.

Typically, SMEs are organisations of limited complexity, but may still depend on task management and collaboration software. When developing custom software for these domains, a few aspects are different from the task in larger companies. First of all, most employed processes are informal and may be modified on a day-to-day basis. Only a fraction of the processes are formally defined. Instead, the process descriptions are present in form of the employees' implicit knowledge. When the processes are captured for requirements engineering, employees will likely fall prey to *process confabulation*. Process confabulation causes domain experts to recount processes not in the way they occur, which is difficult with daily variations, but idealized versions. Thus, if authorization is employed, many restrictions are based on the idealized processes and may be hindering in the execution of day-to-day business. One reason is that employees of SMEs are often unaccustomed to authorization. Typically, most documents are available to a large part of the employees before the implementation of an SME application. On the other hand, with a large amount of data centralized in one application, management will insist on the implementation of fairly strict authorization rules.

One current trend in application development to overcome the problem of fuzzy requirements is employing *Agile development* principles [7]. Agile development focuses on customer needs, implementing in short iterations and allowing modifications of the plan on a regular basis. In Agile development, working applications are preferred over documentation and domain experts are tightly integrated into the process. The focus on constant modification and refinement of requirements makes Agile development suitable for the development of SME applications.

With continuously changing requirements, development environments need to provide an adequate amount of flexibility as well as small development and deployment overhead, as provided by Web applications. *Ruby on Rails*[1] is a current Web development framework that supports Agile development and draws from the meta-programming features of the programming language *Ruby*. Through a plugin architecture, a large community of developers provide other common features, such as authentication, in Rails plugins.

Even with Agile development using Ruby on Rails, implementing security in SME applications remains a challenge, in particular the process-dependent parts of authorization. Flaws in authorization may lead to a loss of efficiency and a lack of acceptance by the end-users, which might even lead to a premature end of the application development. In this paper, we describe Agile security engineering methods to overcome these obstacles. One aspect of our approach is supporting the end-user development of authorization policies. In particular, we introduce an authorization rules *Domain-specific Language* (DSL) for improving authorization policy development. The language is implemented in the `declarative_authorization` Rails plugin. We report on the early feedback of employing the authorization language in a real-world SME application to improve its authorization policy.

[1] http://rubyonrails.org/

2 Agile Security Engineering

Security engineering in traditional software development is a heavy-weight process. For example, the ISO 27001 standard structures security engineering into the well-known four phases of *Plan–Do–Check–Act* which are iteratively applied [1]. The planning phase includes systematic approaches to threat analysis and risk assessment. Also, the security architecture is to be designed before any implementation takes place. Such a security engineering process does not fit well into Agile development processes, resulting in several conflicts.

– Security is difficult to retrofit [5], so that security ideally needs to be considered from the beginning. In Agile development, where having modifications of the plan is common, the functional requirements are by definition not clear at the beginning. Thus, security measures cannot be developed initially in sufficient detail.
– With an anticipated shift in functional requirements, security architectures designed at one point will become obsolete in the course of the project. Redoing security engineering as proposed by the classical iterative models before implementing additional functional requirements is no option, either. The heavy-weight nature of the process makes it impossible to fit into the common 2 to 4 week iteration cycle of Agile development.
– Traditional security engineering implies a good measure of security documentation and specification. In Agile development, this is counter-productive with the application being a moving target, causing a mismatch of documentation and code to an even larger extent than in traditional software projects.
– Security objectives are non-functional requirements and thus hard to test. In Agile development, refactoring is an important aspect to constantly adapt to changing plans. Refactoring relies heavily on testing to ensure that deep changes do not break the application. Missing tests of the security requirements could thus lead to the introduction of vulnerabilities through refactoring.

Reviewing the published work on security in Agile development, a few solutions to the above-outlined problems are proposed. A very general proposal is to increase overall security skills of development teams. Ge et al. argue that in Agile development even more than in other development processes, security awareness is necessary for all team members [10]. To comply with formal requirements of security reviews, a security expert might rotate through programming pairs, thus implicitly reviewing the code. Aydal et al. report on a case study of security through refactoring with good results [3]. Tappenden et al. describe security tests which could be employed to secure refactoring [17]. Instead of the usual user stories that provide requirements in many Agile methodologies, *abuser* or *mis-user* stories may be employed [13], describing unwanted situations which may be tested. This approach might lack the proper completeness, though, as systematic approaches are needed to capture the wealth of attack vectors. An alternative but less concrete approach is imposing *constraints* on every user story.

For security up front, before any development, Ge et al. propose to have experts agree on overall security principles and a high-level security architecture [10]. Still, as indicated above, this might prove either quite complex when a suitable security architecture is to be found or might arrive in rather useless too general principles. It is a good idea, though, to begin with system hardening and penetration tests early in the iteration cycles even if the system is not yet set up in the target environment. Thus, security issues may be tackled early [11]. Lastly, Chivers et al. argue that in Agile development, the team should concentrate on providing *good-enough security* as, in practice, security is not absolute [5]. It is arguably correct that even the systematic approaches of traditional security engineering do not guarantee completeness.

While the listed approaches may not serve as a one-size-fits-all solution, a few points may be worth stressing. At one point during development, a systematic threat analysis and risk assessment should be undertaken to provide a good understanding of security aspects to focus on. With the addition of further features in later iterations, the findings certainly need to be adapted with changes in assets and additions of attack vectors. Thus, key to effective Agile security engineering remains the flexibility in implementing changes in the security architecture. A second aspect is that it is hard to capture security requirements for processes in a single iteration. Because of process confabulation, authorization particularly needs adjustment by domain experts later on. Documents derived directly from the code may come to help in discussions with domain experts while preventing additional overhead and the risk of outdated documents. In the next section we describe ways of tightly integrating domain experts into security engineering and authorization policy development, taking the aforementioned aspects into account.

3 End-User Development of Authorization Policies

In the development of custom SME applications, it is even more important to tightly integrate end users into the development process than in software development for large enterprises. Usually, there are no current documents on the company's processes but only implicit knowledge of the employees. Even if there has been an ISO 9001 certification, those documents often do not reflect the actual processes. In *Human-Computer Interaction* (HCI) research, the growing field of *End-User Development* [12] pushes the barrier even further; not only should end users be integrated into the development process, but in addition end users should take part in the development, adapting the application to their needs [6, 20].

In the domain of security engineering and authorization policy development, there are three potential actors to design and implement authorization: end users, system administrators and developers. One might argue that authorization configuration should be carried out by administrators. On the other hand, domain knowledge is very important for applying the appropriate measure of restrictions. This means that end users are better suited for the task, at least supporting the administrator. Developers also play an important role in the process by having intimate knowledge of the application. With many authorization decisions being based on the application's underlying data model, which may have to be modified to allow specific authorization rules, it is very important to have the developers take active part in the development. Thus, ideally, an authorization policy development would offer the appropriate level of abstraction to each of these actors [8]. Therefore, the following mechanisms are needed:

– An authorization language and data model primarily for developers to implement authorization policies. The language and data model should be simple enough to help end users to discuss and validate the current policy. It might even be possible for them to correct and develop authorization rules using the language.
– Alternative, e.g. graphical, representations of the effective authorization policy concerning specific objects and users to mitigate the complexity of authorization by offering transparency.
– A UI for overcoming barriers posed by textual specifications to some end users.

4 The Declarative Authorization Plugin

For supporting Agile security engineering and end-user integration in authorization rules development, we developed an authorization rules DSL and supporting development tools. We implemented the DSL and the tools as the Ruby on Rails `declarative_authorization` plugin[2], made available under the MIT Open Source license. Currently, we use the plugin in a real-world collaboration and task management Web application that relies on Ruby on Rails as the underlying Web application framework.

The plugin design was guided by the goal of providing the maximum simplicity and readability of the authorization rules DSL and efficient usage of the plugin in Web application development. Other available Rails authorization plugins usually are based on in-line *Access Control Lists* (ACL) of roles, causing redundant authorization rules in program code. In contrast, the `declarative_authorization` plugin separates program and authorization logic, thus offering a *declarative* approach to authorization. The DSL describes the policy for authorization while the application just defines required permissions for specific actions.

4.1 Authorization Rules DSL

The authorization rules DSL was designed for readability and flexibility. The syntax is derived from natural language that can be read in form of sentences, e.g., *role "admin" has permissions on "employees" to "manage."* Symbols beginning with `:`, block delimiters `do`, `end` and hash associations through `=>` remain visible

[2] Available at Github: http://github.com/stffn/declarative_authorization

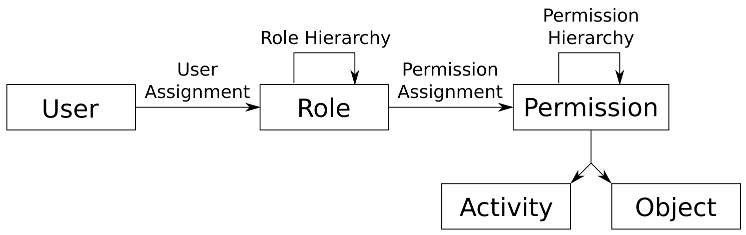

Fig. 1. Role-based access control model

indications that the DSL employs Ruby syntax. We decided to implement the language in Ruby because of Ruby's metaprogramming features, which allow a simple, readable DSL while making use of the benefit of the robust Ruby parser. Also, in the target market of SME applications, applications are increasingly based on Ruby on Rails. A simple example of an authorization rule assigning the permission "manage" on objects of type "employee" to role "admin" is given in the following listing:

```
authorization do
  role :admin do
    has_permission_on :employees, :to => :manage
  end
end
```

The authorization data model behind the DSL is similar to RBAC's. One of many extant variations of the RBAC model is shown in figure 1. The model defines *users*, which are assigned to *roles* in an n:m relation. On the other hand, *permissions* are assigned to roles in an n:m relation as well. Permissions are often described as a combination of *activities* on *objects*. Thus, to evaluate the authorization of a user with respect to a specific object, permissions assigned to the user's roles need to be checked.

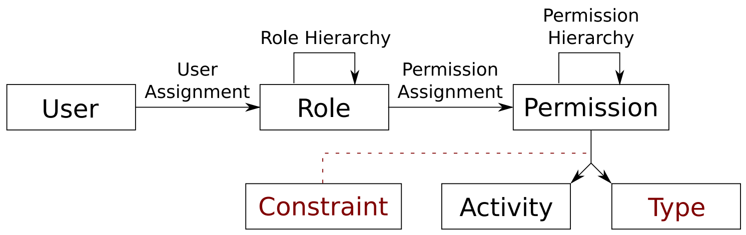

Fig. 2. Authorization rules DSL data model

Instead of defining permissions as activities on objects, the `declarative_authorization` data model (figure 2) uses activities on *types* of objects, such as "employees", to increase maintainability. Permissions on individual objects are realized through context authorization constraints [4]. E.g., for restricting

Listing 1. Example authorization rules

```
 1 authorization do
 2   role :admin do
 3     has_permission_on :employees, :to => :manage
 4   end
 5
 6   role :branch_admin do
 7     includes :employee
 8     has_permission_on :employees, :to => :manage do
 9       if_attribute :branch => is {user.branch}
10     end
11   end
12
13   role :employee do
14     # ...
15   end
16 end
```

"branch admins" to only manage employees of their branch, the statement shown in listing 1 in line 9 is employed. Constraints may be nested for more complex cases. A custom language is used for specifying the constraints so that the same conditions may be used not only to restrict access but also to derive the resulting constraints on database queries. Role hierarchies are realized using the "includes" statement as demonstrated in listing 1 in line 7.

To further improve the usability of the authorization rules language in Agile security engineering, development tools have been implemented. Inside the application, the syntax-highlighted textual representation of the current rules is provided to authorized users. Also, graphical representations have been developed for domain experts to be able to drill down on specific aspects of the authorization rules, as shown in figure 3, while keeping an overview at hand. In the diagram, filled arrows indicate the assignment of permissions to roles, with circles on arrows symbolizing constraints on the assignment. The role hierarchy of "branch admin" including the permissions of "employees" is shown by an unfilled arrow, demonstrating the efficiency of graphical representation for analyzing hierarchical structures.

4.2 Usage in Application Code

In order to support Agile development, ease of implementation in the application is important. Early in the development process, authorization rarely is of high priority. Thus, imposing minimal overhead allows for authorization infrastructure to be integrated early-on, resulting in less refactoring being required later. In Rails, so-called *controllers* are responsible for responding to HTTP requests. Each URI is routed to a controller's *action*. Thus, for a first line of defense, restrictions may be imposed on each action. To enable this with the plugin, only

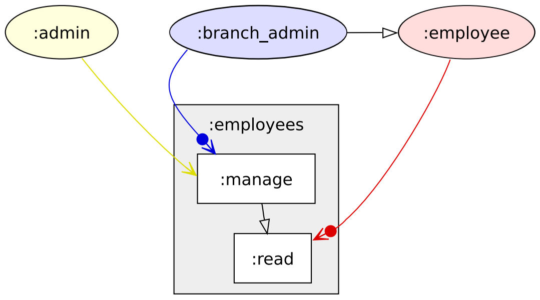

Fig. 3. Graphical representation of authorization rules

a `filter_access_to` statement in a controller is required to cause all requests to that controller to be checked for authorization.

```
class EmployeesController
  filter_access_to :all
  def index
    # ...
  end
end
```

When the "index" action in the `EmployeesController` is called by an HTTP request, the authorization rules are consulted. The `declarative_authorization` plugin considers the roles of the user, which is bound to the current request through separate authentication measures, to decide on allowing the request. If the permissions for "index" have not been assigned to any of the user's roles, the request is denied. If the permission is assigned with additional authorization constraints, objects might be examined to evaluate the constraints.

The "index" action in the `EmployeesController` will provide a list of employees, causing a check of "read" permissions according to a preconfigured mapping. To only display those employees that the current user may read, constraints need to be imposed on a database query for some roles, according to the authorization policy. To enable these automatic constraints, the developer only has to use a `with_permissions_to` call instead of manually constructing the database query conditions, as shown in the following example.

```
class EmployeesController
  filter_access_to :all
  def index
    # @employees = Employee.find(:all, :conditions => ...)
    @employees = Employee.with_permissions_to(:read)
  end
end
```

Thus, with the authorization rules shown in listing 1, users of role "branch admin" would only see the intended list of employees in their branch while minimal extra effort is needed in application development. More importantly, the code does not need to be changed when authorization conditions change, allowing developers to focus on functional and security requirements at different points in time.

5 Early Feedback

In order to evaluate the authorization rules language with respect to its use in Agile security engineering, we employed the `declarative_authorization` plugin in a real-world SME application. The application currently has nine roles and permissions on objects of 35 types. It is a collaboration and task management application that is employed in quality management of automotive parts.

We used the applications authorization rules in discussions with two domain experts. The domain experts use the application regularly as end-users but have not taken part in the programming of the application. In addition to the discussions, we conducted interviews with the domain experts to capture their subjective views on the viability of using textual authorization rules and graphical representations for helping in discussion, finding policy flaws, and allowing end-user modifications.

In both discussions, the textual representation of the authorization rules proved very helpful in improving the current rules. Two flaws within the authorization rules were identified. E.g., an overly narrow restriction on the role of quality inspectors would have prevented their flexible operation for different branches of the SME. The flaws might have hindered the workflow in specific situations by being overly restrictive. In the interviews, the domain experts acknowledged the helpfulness of the textual and graphical representation of the actual authorization rules that are being enforced. Still, for modifications or additions by themselves, both would prefer a user interface.

6 Conclusion and Future Work

When considering custom-built applications for task management and collaboration, Agile development of Web applications helps in efficiently fulfilling SMEs' requirements. To design appropriate security mechanisms, traditional security engineering does not fit well with its heavy-weight processes, though. Agile security engineering processes, as described in this paper, provide an alternative approach by integrating domain experts more tightly into the security engineering process. One important aspect of Agile security engineering is the development of authorization policies. We introduced a tool to support the Agile authorization policy development through a simple and readable authorization rules language and its implementation in the Rails `declarative_authorization` plugin. While certainly not applicable to every kind of application, authorization policies may gain in precision through more intense integration of domain experts and thus improve the effectiveness of the application with only minimal development overhead. Early positive feedback from the evaluation of the authorization language on a real-world SME project demonstrated the potential of our approach.

In addition to broader empirical work, future work will include the development of user interfaces to complement the existing tools. Following the domain experts' suggestions, the UIs should work on a high layer of abstraction, e.g. only allowing the assignment of permissions to existing roles. In another attempt to improve end-user involvement, we will provide measures for test-driven development of authorization rules and an authorization policy development workflow, thus increasing the reliability of authorization policy development.

Taking into account the required flexibility in SME applications' task management, even improved authorization policy development may not prevent occasional missing permissions that degrade efficiency, though. In order to follow the practice of informal processes in SMEs, we will look into a self-regulatory authorization approach that we call *Self-service Authorization.* This mechanism allows end-users to increase their permissions according to certain restrictions on their own while actions are then appropriately audited.

References

1. ISO/IEC 27001:2005. *Information technology – Security techniques – Information security management systems – Requirements.* ISO, Geneva, Switzerland.
2. ANSI INCITS 359-2004. *Role-Based Access Control.* American Nat'l Standard for Information Technology, 2004.
3. Emine G. Aydal, Richard F. Paige, Howard Chivers, and Phillip J. Brooke. Security planning and refactoring in extreme programming. In Pekka Abrahamsson, Michele Marchesi, and Giancarlo Succi, editors, *XP*, volume 4044 of *Lecture Notes in Computer Science*, pages 154–163. Springer, 2006.
4. Elisa Bertino, Elena Ferrari, and Vijay Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Trans. Inf. Syst. Secur.*, 2(1):65–104, 1999.
5. Howard Chivers, Richard F. Paige, and Xiaocheng Ge. Agile security using an incremental security architecture. In Hubert Baumeister, Michele Marchesi, and Mike Holcombe, editors, *XP*, volume 3556 of *Lecture Notes in Computer Science*, pages 57–65. Springer, 2005.
6. Luke Church. End user security: The democratisation of security usability. In *Security and Human Behaviour*, 2008.
7. Alistair Cockburn. *Agile Software Development.* Addison-Wesley Professional, December 2001.
8. Jie Dai and Jim Alves-Foss. Logic based authorization policy engineering. In *The 6th World Multiconference on Systemics, Cybernetics and Informatics*, 2002.
9. David Ferraiolo and Richard Kuhn. Role-based access controls. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.
10. Xiaocheng Ge, Richard F. Paige, Fiona Polack, and Phillip J. Brooke. Extreme programming security practices. In Giulio Concas, Ernesto Damiani, Marco Scotto, and Giancarlo Succi, editors, *XP*, volume 4536 of *Lecture Notes in Computer Science*, pages 226–230. Springer, 2007.

11. Vidar Kongsli. Towards agile security in web applications. In *OOPSLA '06: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 805–808, New York, NY, USA, 2006. ACM.
12. Henry Lieberman. *End user development.* Springer, 2006.
13. John McDermott and Chris Fox. Using abuse case models for security requirements analysis. In *ACSAC '99: Proceedings of the 15th Annual Computer Security Applications Conference*, page 55, Washington, DC, USA, 1999. IEEE Computer Society.
14. Sejong Oh and Seog Park. Task-role-based access control model. *Inf. Syst.*, 28(6):533–562, 2003.
15. Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.
16. Yuqing Sun, Xiangxu Meng, Shijun Liu, and Peng Pan. Flexible workflow incorporated with RBAC. In Weiming Shen, Kuo-Ming Chao, Zongkai Lin, Jean-Paul A. Barthès, and Anne E. James, editors, *CSCWD (Selected papers)*, volume 3865 of *Lecture Notes in Computer Science*, pages 525–534. Springer, 2005.
17. Andrew Tappenden, Patricia Beatty, and James Miller. Agile security testing of web-based systems via httpunit. In *AGILE*, pages 29–38. IEEE Computer Society, 2005.
18. Roshan K. Thomas and Ravi S. Sandhu. Task-based authorization controls (TBAC): A family of models for active and enterprise-oriented autorization management. In *Proceedings of the IFIP TC11 WG11.3 Eleventh International Conference on Database Securty XI*, pages 166–181, London, UK, UK, 1998. Chapman & Hall, Ltd.
19. Jacques Wainer, Paulo Barthelmess, and Akhil Kumar. W-RBAC - a workflow security model incorporating controlled overriding of constraints. *Int. J. Cooperative Inf. Syst.*, 12(4):455–485, 2003.
20. Mary Ellen Zurko and Richard T. Simon. User-centered security. In *NSPW '96: Proceedings of the 1996 workshop on New security paradigms*, pages 27–33, New York, NY, USA, 1996. ACM.