Run-Time Generation, Transformation, and Verification of Access Control Models for Self-Protection

Christopher Bailey
School of Computing
University of Kent, UK
c.bailey@kent.ac.uk

Lionel Montrieux
Centre for Research in Computing
The Open University, UK
lionel.montrieux@open.ac.uk

Rogério de Lemos
School of Computing
University of Kent, UK
CISUC, Coimbra, Portugal
r.delemos@kent.ac.uk

Yijun Yu
Centre for Research in Computing
The Open University, UK
yijun.yu@open.ac.uk

Michel Wermelinger
Centre for Research in Computing
The Open University, UK
michel.wermelinger@open.ac.uk

ABSTRACT

Self-adaptive access control, in which self-* properties are applied to protecting systems, is a promising solution for the handling of malicious user behaviour in complex infrastructures. A major challenge in self-adaptive access control is ensuring that chosen adaptations are valid, and produce a satisfiable model of access. The contribution of this paper is the generation, transformation and verification of Role Based Access Control (RBAC) models at run-time, as a means for providing assurances that the adaptations to be deployed are valid. The goal is to protect the system against insider threats by adapting at run-time the access control policies associated with system resources, and access rights assigned to users. Depending on the type of attack, and based on the models from the target system and its environment, the adapted access control models need to be evaluated against the RBAC metamodel, and the adaptation constraints related to the application. The feasibility of the proposed approach has been demonstrated in the context of a fully working prototype using malicious scenarios inspired by a well documented case of insider attack.

Categories and Subject Descriptors

D.4.6 [**Software Engineering**]: Security and Protection—*Access controls, Verification*

General Terms

Security, Verification

Keywords

Adaptive security, RBAC, model verification, self-adaptation

SEAMS '14, June 2-3, 2014, Hyderabad, India

1. INTRODUCTION

The incorporation of self-adaptability into complex systems inevitably involves the creation and manipulation of several diverse models related to the different viewpoints of the system and its environment. In this paper, we show how this can be achieved in the context of systems security, in particular, how self-adaptive access control (the parametric adaptation of permissions and user access rights) can be an effective solution in protecting against insider threats [13].

Traditionally, organisations rely on audit trails and human administrators to monitor access control systems for the purpose of identifying and handling malicious behaviour, and the consequences have been dire [9, 12, 13]. The application of self-adaptation to access control is a promising solution to improving management of access in organisations, in particular, the timely identification and response to malicious user behaviour for better protecting an organisation's set of resources from internal attacks. However, the risk with self-adaptive access control is the potential negative impact to an organisation, should unnecessary or invalid adaptation be carried out to systems that are of a critical nature. The use of models, together with model transformation and verification at run-time, is seen as a solution to ensure that when handling malicious behaviour, we are able to obtain a verified modelled state of access control that conforms to the organisation's own requirements.

Authorisation infrastructures, which implement access control, are typically deployed as a set of independent systems that conform to an access control methodology, such as Role Based Access Control (RBAC) [36]. For example, an authorisation infrastructure may contain an access control service, which provides access decisions based on a set of access control rules, in addition to an identity service, which maintains a set of users and user assigned access rights. Users use their access rights, which are evaluated by the access control service, to assert whether or not the user should be given access to a resource. When adapting authorisation infrastructures (e.g., the removal of a user's assigned access right to a database), it is necessary to maintain a model of each type of system service to produce a complete model of access that conforms to the system's implemented access control methodology. Model transformation for producing

access control models allows self-adaptive controllers to abstract from implementation specific models, and verify access control as whole, as opposed to individual verification of models relating to independent systems.

The contribution of this paper is related to the provision of assurances for complex self-adaptive systems that require the manipulation of several diverse models. We propose a run-time approach based on model generation, transformation, and verification for providing assurances that the deployed adaptation conforms to the system requirements. This approach is applied, specifically, to self-adaptive access control systems in which adapted access control models are verified before being deployed. The overall goal of our approach is to identify malicious behaviour, in the form of insider attacks, in order to automatically modify policies and rules for mitigating ongoing attacks or prevent future ones. In order to show the feasibility of the proposed approach, we have developed a fully working prototype in which the implementation specific models, including the solutions to deal with the attacks, are transformed into an RBAC model using the Atlas Transformation Language (ATL) [21]. This RBAC model is then verified using the rbacDSML verification tool [1, 27], which is able to verify a UML model of RBAC against OCL constraints. To evaluate our approach, we simulate a documented case of insider attack within our own deployed self-adaptive authorisation infrastructure [4]. The simulation has demonstrated the ability of our prototype to handle multiple attacks using verified adaptations of the authorisation infrastructure.

The rest of this paper is structured as follows. In Section 2, we present background to our work. Section 3 describes our approach of model generation, transformation and verification in the context of a real life insider attack. In Section 4, we describe an experiment involving our prototype, demonstrating the handling of an insider attack. Section 5 discusses related work in terms of access control, verification, and self-adaptation. Finally, Section 6 concludes by summarising the work done so far, and indicates future directions of research.

2. BACKGROUND

2.1 Role Based Access Control (RBAC)

Role-Based Access Control (RBAC) is an access control methodology that features roles as first-class citizens [36]. In RBAC, users are not assigned permissions directly. Instead, they are assigned roles, and those roles are in turn assigned permissions, in order to facilitate the maintenance of large access control policies. RBAC also features role hierarchies, where a role inherits its ancestors' permissions, as well as constraints. Example of constraints are static separation of duties (SSoD), in which two roles cannot be assigned to a user, and dynamic separation of duties (DSoD), where two roles cannot be activated at the same time by a user. RBAC has been standardised by OASIS [30], and is composed of four levels, each one adding new constructs on top of the lower one.

2.2 RBAC Model Verification

Large access control models can be difficult to manage by hand. Ensuring the consistency of the model, but also making sure that the model conforms to the designers' requirements can become a non-trivial and error-prone task when carried out manually. An automated verification mechanism can greatly increase the quality of an access control model, and reduce the number of errors as well as the time spent maintaining access.

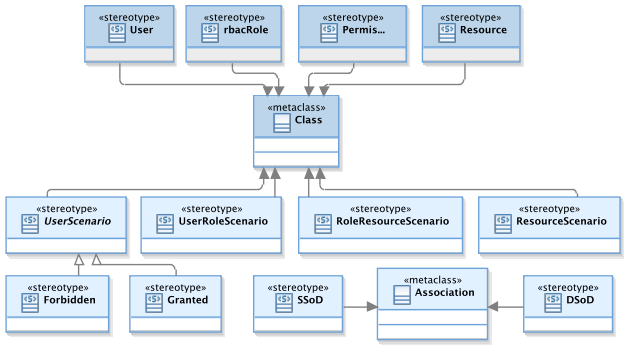

Figure 1: rbacDSML metamodel in MOF

As a domain specific language, and a verification tool, rbacDSML allows one to model RBAC policies that conform to the RBAC standard [30], alongside *scenarios*, which are instantiations of the designers' requirements. rbacDSML is implemented as a UML profile, and ensures the consistency of the policy as well as its satisfaction of the scenarios using OCL constraints [1, 27]. rbacDSML is available as an open-source plugin for IBM Rational Software Architect [20]. A standalone version is also available, currently with a limited set of features.

Since rbacDSML is a UML profile, it is defined as an extension of the UML metamodel that adds new stereotypes and associations to standard UML models. Figure 1 shows the extension of the UML metamodel for rbacDSML, in MOF, the OMG language for meta-model representation [33]. Users, roles, permissions and resources are represented by stereotypes attached to UML classes. Static and dynamic separation of duty constraints are represented as stereotyped associations between two roles. Role hierarchies are represented as class hierarchies. These constructs are a one-to-one translation of the standard RBAC model. In addition to standard RBAC, rbacDSML provides several types *scenarios*, that are represented as stereotypes attached to classes. *User scenarios* require that a user, given a set of active roles, must be able to (resp. not be able to) access a set of resources if stereotyped with *Granted* (resp. *Forbidden*). *User - role scenarios* require a role to be assigned to at least one user. *Role - resource scenarios* require a set of resources to be accessible using the permissions given by a role. Finally, *resource scenarios* require a resource to be accessible by at least one user.

rbacDSML uses seven OCL constraints: two to verify that SSoD and DSoD constraints are not violated; one to verify that roles activated in a scenario have been assigned to the user involved in the scenario; and finally, one constraint for each type of scenario.

2.3 Self-Adaptive Authorisation

Authorisation infrastructures exist to control access to an organisation's electronic resources (e.g., web applications, servers, databases). They are represented as a collection of identity services [25], which maintain information about

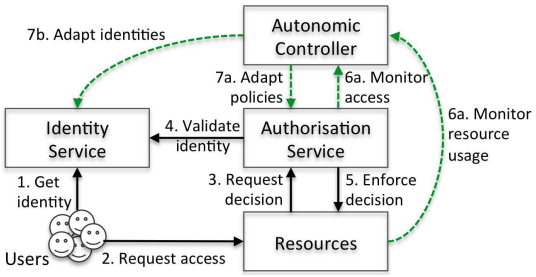

Figure 2: SAAF conceptual design

users and their access rights, and authorisation services [14], which enable access control (the evaluation of user access rights). Management of authorisation infrastructures has increased in complexity, as a result of large user bases and federation [29], but typically rely on human administrators to maintain the conditions of access, and identify when access should change.

Previous work has proposed a Self-Adaptive Authorisation Framework (SAAF) as a solution to managing access control autonomically [3]. SAAF is based on the MAPE-K [23] feedback loop for identifying and responding to insider threats [13], which are typically conducted through user exploitation of legitimately assigned access rights. The provision of self-adaptation to authorisation infrastructures has shown to be a promising solution to respond to insider threats [4, 5], as the adaptation of access control can prevent or limit a malicious user's ability to access protected resources, thus mitigating the attack. The SAAF controller performs adaptations when reacting to identified insider attacks, either by modifying access control policies or user access rights (e.g., the removal of user RBAC roles).

Figure 2 presents a conceptual view of SAAF in which an autonomic controller monitors and adapts multiple systems within an authorisation infrastructure. This presents a challenge since no single system provides a complete view of access in terms of what users own in access rights, what access control rules exist, and finally, how users are utilising access rights. In addition to this, each system may differ on their implementation of the access control methodology that the system claims to conform to (e.g., the RBAC standard). As a result, it can be said that each authorisation service has its own implementation specific model representing part of an access control model (such as, active RBAC roles and permissions). Moreover, an existing limitation of SAAF is its inability to verify an adapted state of access before it is realised in an authorisation infrastructure. In its current form, SAAF assumes that whatever adaptations take place will not break conformance to the service's implemented access control methodology (RBAC), nor conflict with application domain requirements (e.g., ensure access to business critical systems). The use of models (to identify state of access), and model verification at run-time, is seen as a solution to the above existing SAAF limitations.

3. APPROACH

Authorisation infrastructures can be considered critical for organisations, as they provide services to an organisation that enables their users to access resources and perform their duties. A primary concern when adapting a user's access to resources is to obtain guarantees that the resulting adaptation will resolve problems without creating additional ones. It is also important to ensure that any adapted model of access conforms to the authorisation infrastructure's implemented access control methodology (i.e., RBAC).

In the following, we take as an example, a well reported case of IP theft involving a chemical research company [13], where, as any modern organisation requires, there is a need to provide secure access to organisational resources to relevant employees. The chemical company at any given time may need to modify its authorisation infrastructure due to the organisation undergoing natural change (new users, organisational roles, resources), or as a result of access being defined incorrectly, or due to the identification of malicious activities exhibited by their users. As such, the chemical company case study provides the basis to highlighting the benefits of using models, and their run-time verification, in self-adaptive authorisation infrastructures. We do not discuss the detection of insider threats since in a previous work [4] we have shown how the SAAF controller identifies and responds to extreme usage of resources. Such detection techniques can be improved upon with existing approaches, including, anomalous detection through machine learning [10] to handle the detection of unknown attacks.

In this section, we detail our approach in which a SAAF controller makes use of models for capturing the current state of access control, and of model transformation for supporting the verification of those models. Verification enables a controller to identify whether the *adapted* modelled state of access conforms to the authorisation infrastructure's implemented access control methodology (i.e., RBAC), as well as ensuring the model of access meets the application domain's requirements.

3.1 Chemical Company Case Study

In late 2005, a chemical research company was a victim to an insider attack [13] in which an attacker stole around 22,000 sensitive documents from an *ElectronicLibrary*. The attacker, at the time, was an employee of the chemical company, whereby it was assumed he had legitimate access rights to the *ElectronicLibrary*. The attack was committed over a period of 4 months, and only identified once the attacker had ended his contract and begun working for a competitor.

From the account of the attack it is difficult to surmise a complete picture of the system, however it is likely the chemical company operated an authorisation service to manage access to their *ElectronicLibrary*, and utilised identity services to maintain access rights of their users. For the purpose of demonstrating our approach, we make the assumption that the chemical company implements RBAC (due to RBAC's popularity in industry) as their access control methodology, as shown in Figure 3, where users have relevant roles, such as *Researcher*, who have permissions, such as *Get Document from ElectronicLibrary*. To implement RBAC, we provide an identity service referred to as LDAP [25], which is a directory service commonly used to hold information (including user roles) about users within an organisation, and an authorisation service known as PERMIS [14], a standalone service used to generate RBAC access control decisions based on roles owned by users. It is assumed the attacker would have utilised his role stored within the *LDAP* directory for gaining access to the *ElectronicLibrary*, whereby the *ElectronicLibrary* would request

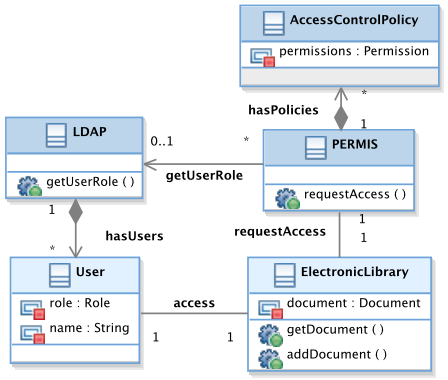

Figure 3: Interpreted chemical company class diagram

an access control decision from the *PERMIS* authorisation service. The *PERMIS* authorisation service would in turn check permissions held within its own access control policy, to see if the accessing user has the relevant roles for access.

Although the identification of the attack is not clearly discussed in the account, we will classify the properties of the attack in terms of the attacker's usage of the *ElectronicLibrary*. Notably, the attacker's usage was 15 times higher than the next highest user downloads, and many of the downloaded documents were not related to the attacker's role [13]. Had a SAAF-like controller been deployed in this case, it would have been possible for the attack to be detected (and thus responded to) using a set of predefined rules that enable the controller to detect unacceptable patterns of user behaviour [28].

3.2 Model Generation

Before the SAAF controller can begin to identify and respond to the insider attack the chemical company was victim to, it must generate a set of implementation specific models representing the active identity services, and authorisation services within the authorisation infrastructure. The SAAF controller generates these models via its monitor component within the MAPE-K loop, whereby probes obtain information about the services, which is in turn injected into a model of the service. A model generated from the LDAP identity service provides a view of active users and user role assignments. A model generated from the PERMIS authorisation service provides a view of active RBAC access control rules, such as the permissions of each role. Both models generated conform to implementation specific metamodels, which describe what can exist in terms of LDAP and PERMIS. The PERMIS metamodel is automatically generated from PERMIS's own proprietary access control policy schema, whereas the LDAP metamodel has been defined solely to capture the necessary user information in terms of how LDAP views a user's set of assigned roles. A benefit in utilising implementation specific models is that a service's model can be adapted and easily realised against the deployed service. However, a limiting factor is that the SAAF controller must maintain an understanding of each type of service it is to control, in order to identify how the service implements RBAC.

3.3 Model Transformation

Relying on implementation specific models is useful since it is easier to understand and adapt the current state of the LDAP and PERMIS services, considering that each model is independent to each other and is subject to independent change. However, for verifying a modelled state of access, the SAAF controller combines the implementation specific models into a single model, which embodies the authorisation infrastructure's access control methodology (RBAC). Figure 4 portrays the use of models within the SAAF controller as a set of model to model transformations, conforming to the model driven engineering framework [11]. The transformation diagram describes two domains, the first is the SAAF controller's domain, where it maintains metamodels of the LDAP identity service (LDAPUser) and the PERMIS authorisation service (PERMIS). The RBAC metamodel is an implementation of the RBAC standard, relevant to SAAF. The second domain is RBACDSML [27], which maintains a metamodel to model RBAC in UML for the purpose of verification, which is described in Section 3.5.

The RBAC model represents the SAAF controller's knowledge of the current state of access. It combines a view of the active RBAC access control rules maintained in the PERMIS authorisation service, as well as current user role assignments held in the LDAP directory. In the chemical company's case it represents a combined view of the active RBAC access control rules maintained in the PERMIS authorisation service, as well as active user role assignments held in the LDAP directory. The RBAC model is produced as a result of an Atlas Transformation Language (ATL) [21] transformation program, referred to as *PERMIS+USER2RBAC*. The *PERMIS+USER2RBAC* transformation takes as input a model generated from what is injected from the LDAP directory, and a model generated from what is injected from the PERMIS active access control policy. Each time the implementation specific models are updated, via injection of data collected by probes in the authorisation infrastructure, the SAAF controller performs the *PERMIS+USER2RBAC* transformation. This ensures that the SAAF controller maintains an up-to-date modelled state of access that exists within the authorisation infrastructure. Once the implementation specific models have been transformed into the RBAC model, the SAAF controller is able to reason about the state of access and adapt the RBAC model in light of detected attacks.

Whenever the RBAC model undergoes adaptation, the RBAC model must be transformed back into the implementation specific models, and then deployed back into the relevant services of the authorisation infrastructure. Transformation programs are beneficial here, as the SAAF controller is not concerned with how to adapt implementation specific models, rather, relies on the transformation program to transform the changes made against the RBAC model, into the relevant LDAP or PERMIS model. Two separate transformation programs have been created to enable this: 1) *RBAC2PERMIS*, which creates a new PERMIS access control policy model, capturing new RBAC access control rules, and 2) *RBAC2USER*, which creates a new user model specifying new user role assignments.

3.4 Adaptation

Figure 5 depicts a partial view of the SAAF controller capturing the process of analysis, verification and selection

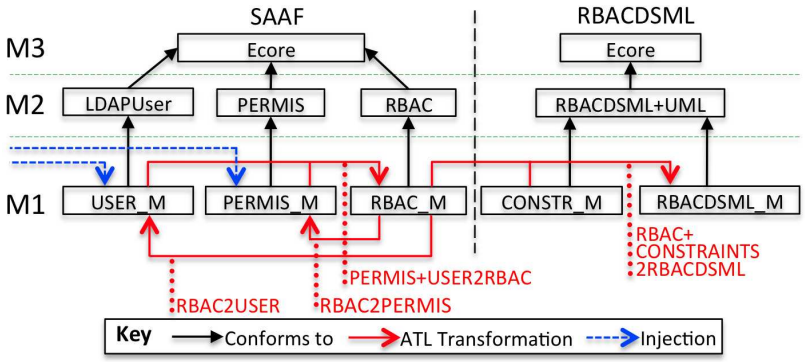

Figure 4: Model transformation in SAAF

of a verified RBAC model. The SAAF controller comprises the analysis, which generates new access control models (in terms of RBAC), and the planning, which selects the most appropriate access control model amongst the valid ones. For each identified attack, and depending on the type of attack and current access control model, obtained by inspecting the authorisation infrastructure, the SAAF controller selects a subset of solutions applicable to a particular attack. The solutions are tailored and incorporated into the current access control model. In order to ensure that the adapted access control models are relevant for the user identified as malicious and the resources they are accessing, each adapted RBAC model is verified against the the rbacDSML verification tool. Depending on a positive output of the verification tool, the adapted RBAC models are collated into a set of verified adapted RBAC models, applicable for planning.

At deployment-time, the SAAF controller is loaded with a set of predefined solutions to respond to malicious events. The solutions match a finite set of actions that can be performed within the application domain, and are parametric in order to tailor the solutions to specific cases of insider attacks. Given a detected attack, a solution is selected from the following alternatives: 1) increasing, limiting or removing access rights owned by an individual, 2) increasing or limiting the scope of access defined by RBAC access control rules, 3) warning the individual(s) of their behaviour, and 4) monitoring the behaviour further. Associated with each solution is an impact, since depending on the type of action invoked could cause either negligible or severe consequences to the system (which may be warranted given the severity of the attack detected). Which solution is selected depends on how severe the SAAF controller deems the identified behaviour to be, in terms of utility, for example: the number of non malicious users impacted negatively by the solution (thus losing access to resources). This may be necessary for cases where many users are identified as being malicious in relation to specific resources or roles, where changing RBAC access control rules provide a more effective means to responding to the attack.

A simple case of adaptation can be made in terms of the RBAC metamodel (Figure 1). Adaptation can be performed against an RBAC model in terms of removing associations from users and roles, resources and permissions, and roles and permissions. For example, removing the role *Researcher* from user *Bob* to prevent *Bob* from accessing resource *ElectronicLibrary*.

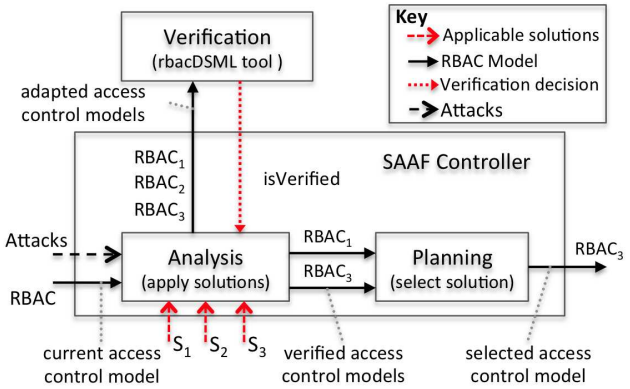

Figure 5: SAAF controller model verification

Before the state of access control can be adapted to handle an attack, solutions applicable to the problem are verified for their conformity to the RBAC standard, as well as, against adaptation constraints. Adaptation constraints represents the application domain's mandatory requirements for access that should be maintained when responding to an attack. For example, the chemical company may specify that access to the *ElectronicLibrary* must be maintained for a specific role, regardless of identified attacks, or that each resource should be accessible by at least one user. In order to perform verification, the application's adaptation constraints are modelled at development-time as scenarios, in conformance to rbacDSML (Figure 1) and captured by the *CONSTR_M* model (Figure 4). For the SAAF controller to make use of the rbacDSML verification tool, which is capable of verifying RBAC in UML with OCL constraints, the adaptation constraints have to be transformed into a UML model of RBAC by invoking *RBAC+CONSTRAINTS2RBACDSML*.

The final part of the adaptation engine is to select an optimum verified RBAC model to be deployed in the target authorisation infrastructure. This differs to previous work [4, 5] whereby all solutions were assumed to result in a verified state of access, and adaptation constraints were not considered.

3.5 Verification

The purpose of the verification step is twofold: to make sure that the adapted RBAC model is consistent, i.e. that

it conforms to the RBAC standard and that separation of duty constraints are not violated, and to make sure that the RBAC model does not violate the adaptation constraints. To this end, a transformation combines the adapted RBAC model with the application's adaptation constraints into an rbacDSML model. The standalone version of the rbacDSML tool's OCL constraints carry out the verification process.

Each OCL constraint has been defined on the rbacDSML profile metamodel. Each constraint has a *context*: a stereotype on which the constraint applies. The verification process will evaluate each OCL constraint on *every* instance of its context stereotype present in the model. For example, the SSoD constraint's context is the *User* stereotype. If there are 20 users in the model, this constraint will be evaluated 20 times, once for each user.

rbacDSML returns a list of violated constraints, together with their context elements. Therefore, if the SSoD constraint has been violated for user *Bob*, then rbacDSML will return (WF SSoD, Bob) as an element of the list. If no constraints have been violated, rbacDSML returns an empty list.

If the verification has succeeded (i.e., an empty list is returned), the adapted RBAC model is acceptable, and it will be deployed. If one or several errors are detected, the RBAC model is not acceptable, and the SAAF controller will select another candidate RBAC model. If the SAAF controller runs out of candidates, the adaptation is simply cancelled, and the state of access remains the same. The failure to handle an adaptation is simply logged by the SAAF controller, however this could be improved with a mail notification to human administrators.

4. EXPERIMENTS

In this section, we evaluate our approach through a scaled simulation inspired by the chemical company insider attack within an RBAC self-adaptive authorisation infrastructure. A SAAF controller is deployed in order to identify the insider attack, and attempt to handle the attack by adapting the state of access. We focus our evaluation on verification, by demonstrating that the SAAF controller considers several solutions while handling the simulated attack, whereby only a verified solution is selected. To showcase the evaluation of complex solutions, we extend the insider attack scenario to consider the case of multiple collaborating attackers. This enables the demonstration of solutions involving adaptations against access control policies. Evidence is provided by way of a table in which we capture the escalation of the attack and the increasing set of solutions relevant to resolving the attack. For each phase of the attack, the available solutions are verified, restricting the overall set of solutions for the SAAF controller to select, and deploy. Finally we provide scalability measures in regards to the verification of access control models at run-time.

4.1 Scenario Setup

To simulate the insider attack, we deploy an RBAC authorisation infrastructure containing an LDAP directory with several users, a PERMIS authorisation service with an access control policy, and a SAAF controller. We refer to this deployment as a self-adaptive authorisation infrastructure and is based on Figure 3. The authorisation infrastructure's state of access (pre-adaptation) is identified in Figure 6. At deployment-time, 8 users are active, with assigned roles held

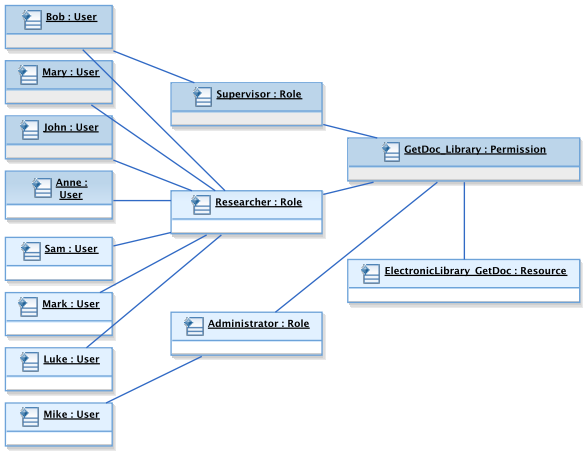

Figure 6: Deployed state of access (RBAC)

in the LDAP directory. The PERMIS authorisation service has one active access control policy, which states users with the role *Supervisor*, *Researcher* and *Administrator* can access the *ElectronicLibrary_GetDoc* resource. Note in this case, multiple roles have access to the same permission expressed in the PERMIS access control policy.

The properties of the chemical company case suggest a long term attack, whereby a single user downloaded a high volume of documents over a period of 4 months. In addition, the detection of the attack suggests it was made through calculating the deviation of the attacker's historic usage of the *ElectronicLibrary* to other users, where the attacker's usage was 15 times greater [13]. For the purpose of the paper, we scale down the attack and simulate usage of the *ElectronicLibrary* within a period of 4 hours (as opposed to the 4 months in which the attack was conducted), and instead of 22,000 documents downloaded, we assume 240. We also assume that the acceptable number of downloaded documents within that period of 4 hours is 16, which is 15 times less than 240.

4.2 Deployment

The self-adaptive authorisation infrastructure is hosted across two virtual machines, each with 1024MB of RAM and running Ubuntu 12.04.3 LTS. The virtual machines represent an identity service, containing an *LDAP* directory, and an authorisation service, containing a deployment of *PERMIS*. The SAAF controller is also deployed on the authorisation service machine, where it is best suited to managing PERMIS access control policies. Finally, the existence of the *ElectronicLibrary* is simulated via access requests made in the form of HTTP requests from a Windows 7 machine (2GB of RAM). Access is simulated through the use of an automated script, sending access requests to *PERMIS*, whereby user access rights are evaluated. Each granted request to download a document from the *ElectronicLibrary* is seen as synonymous with a user downloading a document.

4.3 Application Domain Requirements

The application domain represents the victim organisation (the chemical company), whereby the organisation owns the authorisation infrastructure, its deployed services, the

SAAF controller, and the protected resources (i.e., *ElectronicLibrary*). The application domain's requirements are necessary in governing the extent of adaptation, regardless of what malicious behaviour is detected, and are modelled as rbacDSML scenarios (adaptation constraints) as described in Section 2.2. In these cases, it may be that the chemical company is only willing to risk automated adaptation where only low level workers are impacted. To reflect these concerns, we deploy the following adaptation constraints:

- C1 *Administrator* role must maintain access to all resources (*Role Resource Scenario*)
- C2 At least one user must be assigned the *Administrator* role (*User Role Scenario*)
- C3 Each resource should be accessible by at least one user (*Resource Scenario*)

4.4 Identification

The SAAF controller is capable of identifying malicious behaviour within an authorisation infrastructure, either via signature, pattern, or deviation based behaviour rules. Behaviour rules denote pre-defined conditions which represent malicious behaviour within the target authorisation infrastructure. Signature based refers to access or usage of a resource from blacklisted subjects or IP addresses. Pattern based refers to a pattern of usage of access or a resource, conducted by a user over time, for example, the rate of access requests to a resource in a given period of time. Finally, deviation based refers to a user's pattern of usage in comparison to historical usage of a user or other users, for example, how one user's activity compares to that of another user with the same role.

Considering the properties of the attack, the SAAF controller is deployed with a single deviation type behaviour rule, whereby should it detect usage of the *ElectronicLibrary* from users with the role of *Researcher* as greater than 3 times the frequency of average number of downloads (within the 4 hour interval), malicious behaviour is detected. We assume for the purpose of the experiment that the average number of downloads is considered to be at most 16, as discussed in the adaptation scenario. In addition, to classify severe behaviour, a composite rule is applied [4], which indicates after the deviation rule has been broken multiple times, the behaviour is severe enough to warrant adaptations to policies (which generate greater impact).

4.5 Solutions

The SAAF controller is deployed with a set of solutions, tailorable to detectable attacks. All of the below solutions are considered to be capable in resolving malicious behaviour detected by the deviation type behaviour rule expressed in Section 4.4. These solutions are expressed in XML and parsed into the SAAF controller once initiated. Solution *S1* indicates adaptation to the individual, where as, solutions *S2* to *S5* indicates policy adaptation, impacting many individuals. They remain fixed throughout run-time.

- S1 Remove all roles from <user>
- S2 Remove <permission> from <role>
- S3 Remove all permissions to <resource>
- S4 Remove all permissions from <role>
- S5 Remove all permissions from all roles

4.6 Execution

To demonstrate the flexibility of adaptation and verification, we simulate the properties of the chemical company insider attack as a coordinated attack between 4 users with the *Researcher* role, with the intent to carry out IP theft against the *ElectronicLibrary*. There are 4 stages of the attack, capable of demonstrating simple adaptation against individual users, followed by adaptation against access control policies. In each stage a new user is simulated to break the SAAF controller's behaviour rules, allowing SAAF to identify the malicious behaviour and respond to it accordingly. All but three users of the *Researcher* role and the one user owning the *Administrator* role take part in the attack. As each stage is simulated, the number of solutions applicable to the behaviour increases, identifying that the *ElectronicLibrary* is under persistent attack.

The first stage demonstrates the user *Anne* breaking the deviation type behaviour rule by downloading a high number of documents at the start of the 4 hour attack period, using her assigned *Researcher* role. The second and third stage introduce users *John* and *Mary* carrying out similar activity to stage one, again within the same 4 hour window and using the *Researcher* role. Finally, the forth stage simulates the user *Bob* breaking the same behaviour rule, using his *Researcher* role. Each stage considers a set of solutions, whereby the set of verified solutions is captured, and the result of the adaptation engine is shown in terms of a selected verified solution.

4.7 Scenario Results

We have simulated the attack over a period of 4 hours, in accordance to the 4 stages described in Section 4.6. A set of solutions ({*S1*, *S2*, *S3*, *S4*, *S5*}) were deployed in the SAAF controller, relevant to handling the deviation based behaviour rule. The solutions were chosen to demonstrate the verification of invalid and valid RBAC models at runtime. To gain a performance average for the response to each attack stage, the experiment was executed 30 times. For practical reasons, performance averages were obtained from simulating the attack in a reduced attack period of 5 minutes, where adaptation and verification results showed negligible difference to the 4 hour simulation.

Table 1 portrays the 4 stages of attack. In the first two stages, the SAAF controller considers the malicious behaviour to be minor, only identifying solution *S1* as a relevant solution. At this point solution *S1* has been tailored to the role the user is activating (*Researcher*), and the resource they are accessing (*ElectronicLibrary*). In both stages, the tailored solutions result in a verified RBAC model since there is no conflict with the 3 constraints described in Section 4.3. Solution *S1*, which removes *Anne* and *John*'s access rights, thus their ability to access the *ElectronicLibrary*, is chosen as it is the only valid solution available. The solution is realised by transforming the adapted RBAC model into a LDAP user model, which is then used to update the current state of access rights within the LDAP. The adaptation, from detection to response within the authorisation infrastructure, took an average of 18.70 seconds to complete in the first stage, and a average 10.74 seconds to complete in the second stage. The difference in time is assumed to be as result

Table 1: Verification and adaptation results

Stage	User	Identified Solutions	Valid Solutions	Selected Solutions	Avg. Response Time (sec)	Standard Dev.
1	Anne	S1	S1	S1	18.70	1.11
2	John	S1	S1	S1	10.74	0.64
3	Mary	S1, S2, S3, S4, S5	S1, S2, S4	S1	45.12	1.30
4	Bob	S1, S2, S3, S4, S5	S1, S2, S4	S4	44.79	1.31

of the Java virtual machine warming up, in which both the SAAF controller and verification tool runs on.

Once the third stage of the attack was executed, the SAAF controller identified that there was a continuous malicious activity regarding the role of *Researcher*, and the resource *ElectronicLibrary*. As a result, the SAAF controller selects a more severe set of solutions ({*S1*, *S2*, *S3*, *S4*, *S5*}). In this case, the SAAF controller builds multiple RBAC models in accordance to the tailored solutions, and verifies them using the rbacDSML tool, resulting in solutions *S3* and *S5* verified as invalid. This is due to the fact that solution *S3* removes all access to the *ElectronicLibrary*, violating adaptation constraints: *C1*, and *C3*. The same violation of constraints *C1* and *C3* occurred when the SAAF controller deactivated all permissions within the RBAC model, for solution *S5*. Solution *S1* is ultimately chosen as the solution with the optimum utility, given the severity of the attack and the solutions available (Section 3.4). This is a result of the SAAF controller calculating that stronger adaptations (*S2* and *S4*) would cause greater impact than allowing the attack to continue, whereas *S1* does not, as it only impacts the attacker. As with stage 1 and 2, *Marys*'s access right to the *ElectronicLibrary* has now been removed, preventing her from further access. The adaptation of stage 3, from detection to response, took a total average of 45.12 seconds. This is due to additional RBAC models undergoing adaption, transformation into a rbacDSML UML model, and verification using the rbacDSML verification tool.

Finally, in the last stage of the attack, the same solutions are identified and verified similarly to stage 3; however, solution *S4* is selected. Solution *S4* removes all permissions from the *Researcher* role, preventing any future user with the same role from activating it. The solution is realised by transforming the adapted RBAC model into a new PERMIS model, which is then deployed as a new access control policy. This has a negative consequence on the remaining 3 users with the *Researcher* role, as they are no longer capable of accessing the *ElectronicLibrary*. However, the SAAF controller has selected this solution due to the persistent attacks against the *ElectronicLibrary*. A contributing dimension of utility to this is that in stage 4, there are more malicious users that own the *Researcher* role when compared to non-malicious users.

4.8 Scalability of Verification

To demonstrate the scalability of verification in terms of RBAC models, we randomly generated models with up to 10000 elements (where an element could be of type subject, role, permission, and constraint scenario). Each model increased in size, and verified on the same environment described in Section 4.2. For consistency, each model generated contained a ratio of 50% subject elements, 15% role elements, 10% permission elements, 10% resource elements, and 15% constraint scenario elements. The verification of each model was repeated 10 times to obtain an average and standard deviation. The scalability results are shown in Figure 7, where as the model size increased, the verification times were shown to follow a linear pattern. Note that these performance measures only capture the time it takes to complete a verification cycle, and does not represent a complete adaptation cycle (as shown in Table 1).

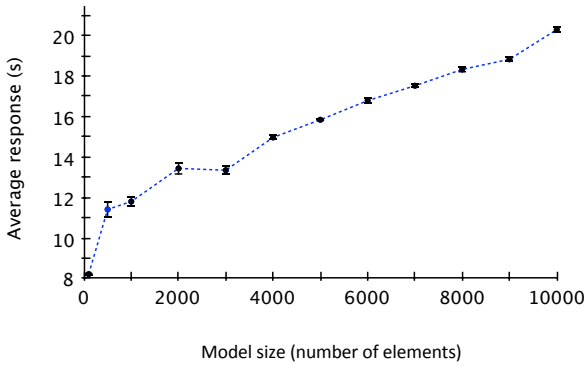

Figure 7: Scalability of model verification

4.9 Discussion

The results have enabled us to demonstrate the verification of RBAC models, which has prevented invalid tailored solutions from being deployed in the target authorisation infrastructure. In addition, we have shown the escalation of attack to be met with the verification and application of stronger solutions, ultimately stopping the collaborated attack through an adaptation to the access control policy.

One restriction in our approach is that verification is confined to mandatory constraints that must always be verified per adaptation. In some cases it can be argued that different levels of verification are needed, given the attack or solutions available. For example, given a minor attack on the *ElectronicLibrary*, the application domain may require a constraint guaranteeing at least one researcher maintains access to the resource. However, should the attack continue and becomes severe, the application domain may require that the constraint is no longer applicable since the *ElectronicLibrary* has suffered a severe attack. One solution to this is to classify identified attacks against available constraints, indicating which constraints should be verified per attack (in addition to mandatory constraints).

In regards to performance, solution verification is a resource intensive operation, and as a result, it takes longer when processing multiple solutions. As SAAF serialises the solution analysis when reacting to attacks, the set of solutions applicable to each attack can be verified in parallel (as each solution is independent of one another).

5. RELATED WORK

5.1 Access Control and Verification

Role Based Access Control (RBAC) is arguably one of the most researched access control models; however, recently, Attribute-Based Access Control (ABAC) models [40] have been receiving more interest from the research community. Sandhu argues that the move from role-based to attribute-based access control opens up new possibilities, as well as new research challenges [35]. RBAC models can be represented using ABAC models, where the roles are defined as attributes. One of the best known ABAC policy languages, XACML [31], features a profile to represent RBAC models [32]. The focus of this paper is on RBAC, both for its standardised and well-understood model, and for its support by PERMIS.

The conformance of XACML policies to designers' requirements has received a lot of interest. Fisler et al. transform access control policies into decision diagrams that can be queried [17]; Hughes and Bultan use a SAT solver on XACML policies to verify that the policy conforms to some properties [19]. Gofman et al. verify properties of RBAC and ARBAC (another RBAC extension) models using RBAC-PAT [18]. The Ponder2 framework also provides policy verification capabilities, using event calculus [6].

Many approaches have also been proposed that allow one to model authorisation policies, and sometimes verify them against requirements, as part of a Model-Driven Engineering approach. rbacDSML is one of them. A few of these approaches use UML profiles to represent RBAC policies, and often query then using OCL constraints; they include UMLsec [22], SecureUML [8], Shin and Ahn's approach [2], and Cirit and Buzluca's UML profile [15]. Kim et al. represent RBAC policies as UML patterns that are then instantiated on a model, using UML template diagrams [24]. Song et al. use Aspect-Oriented Modelling to represent RBAC policies as crosscutting concerns in a UML model, and provides support for verifying properties that the model should satisfy [38]. Sun et al. translate UML models to Alloy in order to verify properties using a SAT solver [39]. Finally, Sohr et al. concentrate on the satisfaction of constraints such as separation of duty, using OCL [37], as well as dynamic, time-based constraints [26], using a domain-specific modelling language (DSML) for RBAC. rbacDSML was very well suited for this paper, because of its open source implementation, and because we were able to define a partial rbacDSML model to represent the application's adaptation constraints, and to merge it at run-time with the rest of the RBAC model in order to form and verify a complete rbacDSML model.

5.2 Self-Adaptive Access Control

To the best of our knowledge model driven approaches have not been used to enable adaptation of authorisation infrastructures, specifically in the establishment of security concerns and assignment of access to better protect an organisation's resources. The concept of modelling authorisation, specifically security, is not new [7]. System designs and security concerns can be modelled together, to create authorisation infrastructures. The use of metamodels, in this paper, can be likened to efforts in security requirements engineering, whereby models of systems are expressed to refine and generate security policies [16]. However, in our specific case the metamodel is used to aid run-time generation of a model view of an authorisation infrastructure, relating existing models of security concerns and subject privileges.

The concept of utilising model driven approaches [11] to enable self-* properties is not new. Works such as [34] outline a model driven approach in enabling self-managing systems, particularly at an architectural level, whereby the role of models is discussed both during design time and run-time of a system. Whilst our work applies model driven approaches in order to achieve self-adaptive properties, the purpose of our models differ. Rather than modelling architectural state and properties, we focus on modelling volatile parameters of structural components that control component execution. For example, modelling a security concern used to govern authorisation decisions executed by an authorisation service.

6. CONCLUSIONS

In this paper, we have presented the first model-driven self-adaptive approach to access control, capable of handling multiple insider attacks whilst maintaining user-defined application constraints. Assurances that the adaptations to be deployed are valid are obtained through the usage of models, model transformation, and model verification at run-time. We describe the generation of implementation specific models of multiple deployed systems that implement different aspects of access control. These implementation specific models are transformed into a single model of access control, whereby adaptation is carried out and evaluated using model verification. We have demonstrated our approach by deploying a self-adaptive authorisation infrastructure in which we simulate and respond to a well documented case of insider attack against a chemical research company [13], whilst considering several application constraints. The simulation has captured the process of model verification, whereby only verified models are used to adapt the current state of access control within the authorisation infrastructure.

From the evaluations, we have shown that we are able to prevent the simulated attack from continuing, and handling the attack in a more timely manner if compared with approaches that rely on human administrators. There are some limitations with this approach, notably that model verification at run-time is a time consuming operation. The time it takes for our self-adaptive authorisation infrastructure to respond to attack is dependent on the number of solutions that must be verified, as each insider attack could be resolved by multiple solutions. Despite the time it takes to verify adaptations, we consider this to be faster than traditional human based approaches.

Our future work involves the further development of the Self-Adaptive Authorisation Framework (SAAF), specifically, with an aim to improve the performance of verification. As we have currently adopted a brute force approach to verifying all applicable solutions to an attack, we aim to improve upon this by using change impact analysis to only verify a subset of the adapted access control model, depending on the changes made compared to the previous model.

7. ACKNOWLEDGEMENTS

Co-financed by the Foundation for Science and Technology via project CMU-PT/ELE/0030/2009 and by FEDER via the "Programa Operacional Factores de Competitivi-

dade" of QREN with COMPETE reference: FCOMP-01-0124-FEDER-012983.

8. REFERENCES

[1] rbacDSML tool, 2009-2014. http://computing-research.open.ac.uk/rbac/ [accessed January 2014].

[2] G.-J. Ahn and M. E. Shin. Role-Based Authorization Constraints Specification Using Object Constraint Language. In *Proceedings of the 10th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, WETICE '01, pages 157–162. IEEE Computer Society, 2001.

[3] C. Bailey, D. W. Chadwick, and R. de Lemos. Self-adaptive authorization framework for policy based rbac/abac models. In *Proceedings of the 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, DASC '11, pages 37–44, Washington, DC, USA, 2011. IEEE Computer Society.

[4] C. Bailey, D. W. Chadwick, and R. de Lemos. Self-adaptive federated authorization infrastructures. *Journal of Computer and System Sciences*, 2014.

[5] C. Bailey, D. W. Chadwick, R. de Lemos, and K. W. S. Siu. Enabling the autonomic management of federated identity providers. In *Proceedings of the 7th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security: Emerging Management Mechanisms for the Future Internet - Volume 7943*, AIMS'13, pages 100–111, Berlin, Heidelberg, 2013. Springer-Verlag.

[6] A. K. Bandara, E. C. Lupu, and A. Russo. Using event calculus to formalise policy specification and analysis. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, POLICY '03, pages 26–. IEEE Computer Society, 2003.

[7] D. Basin, M. Clavel, and M. Egea. A decade of model-driven security. In *Proceedings of the 16th ACM symposium on Access control models and technologies*, SACMAT '11, pages 1–10. ACM, 2011.

[8] D. Basin, J. Doser, and T. Lodderstedt. Model driven security: From UML models to access control infrastructures. *ACM Trans. Softw. Eng. Methodol.*, 15(1):39–91, Jan. 2006.

[9] BBC. Credit card details on 20 million south koreans stolen. BBC, January 2014. http://www.bbc.co.uk/news/technology-25808189 [accessed January 2014].

[10] E. Bertino, A. Kamra, E. Terzi, and A. Vakali. Intrusion detection in rbac-administered databases. In *Proceedings of the 21st Annual Computer Security Applications Conference*, ACSAC '05, pages 170–182, Washington, DC, USA, 2005. IEEE Computer Society.

[11] J. Bézivin. Model driven engineering: An emerging technical space. In R. Laemmel, J. Saraiva, and J. Visser, editors, *Generative and Transformational Techniques in Software Engineering*, volume 4143 of *Lecture Notes in Computer Science*, pages 36–64. Springer Berlin Heidelberg, 2006.

[12] R. Booth, H. Brooke, and S. Moriss. Wikileaks cables: Bradley manning faces 52 years in jail. The Guardian, 30 November 2010.

[13] D. M. Cappelli, A. P. Moore, and R. F. Trzeciak. *The CERT Guide to Insider Threats: How to Prevent, Detect, and Respond to Information Technology Crimes*. Addison-Wesley Professional, 1st edition, 2012.

[14] D. Chadwick, G. Zhao, S. Otenko, R. Laborde, L. Su, and T. A. Nguyen. Permis: A modular authorization infrastructure. *Concurr. Comput. : Pract. Exper.*, 20(11):1341–1357, Aug. 2008.

[15] Ç. Cirit and F. Buzluca. A UML profile for role-based access control. In *Proceedings of the 2nd international conference on Security of information and networks*, SIN '09, pages 83–92. ACM, 2009.

[16] R. Craven, J. Lobo, E. Lupu, A. Russo, and M. Sloman. Policy refinement: Decomposition and operationalization for dynamic domains. In *Proceedings of the 7th International Conference on Network and Services Management*, CNSM '11, pages 115–123, Laxenburg, Austria, Austria, 2011. International Federation for Information Processing.

[17] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies. In *Proceedings of the 27th international conference on Software engineering*, ICSE '05, pages 196–205. ACM, 2005.

[18] M. Gofman, R. Luo, A. Solomon, Y. Zhang, P. Yang, and S. Stoller. RBAC-PAT: A Policy Analysis Tool for Role Based Access Control. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 5505 of *Lecture Notes in Computer Science*, pages 46–49. Springer, 2009.

[19] G. Hughes and T. Bultan. Automated verification of access control policies using a sat solver. *Int. J. Softw. Tools Technol. Transf.*, 10(6):503–520, Oct. 2008.

[20] IBM. Rational Software Architect 8.0.4, 2012.

[21] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. Atl: A model transformation tool. *Sci. Comput. Program.*, 72(1-2):31–39, June 2008.

[22] J. Jürjens. *Secure Systems Development with UML*. Springer-Verlag, 2005.

[23] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, Jan. 2003.

[24] D.-K. Kim, I. Ray, R. France, and N. Li. Modeling Role-Based Access Control Using Parameterized UML Models. In M. Wermelinger and T. Margaria-Steffen, editors, *Fundamental Approaches to Software Engineering*, volume 2984 of *Lecture Notes in Computer Science*, pages 180–193. Springer Berlin Heidelberg, 2004.

[25] V. Koutsonikola and A. Vakali. Ldap: Framework, practices, and trends. *IEEE Internet Computing*, 8(5):66–72, Sept. 2004.

[26] M. Kuhlmann, K. Sohr, and M. Gogolla. Comprehensive Two-Level Analysis of Static and Dynamic RBAC Constraints with UML and OCL. In *Proceedings of the 2011 Fifth International Conference on Secure Software Integration and Reliability Improvement*, SSIRI '11, pages 108–117. IEEE Computer Society, 2011.

[27] L. Montrieux. *Model-Based Analysis of Role-Based Access Control*. PhD thesis, The Open University, 2013.

[28] H. M. Moore. Andrew and M. David. A pattern for increased monitoring for intellectual property theft by departing insiders. Technical Report CMU/SEI-2012-TR-008, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2012.

[29] R. L. Morgan, S. Cantor, S. Carmody, W. Hoehn, and K. Klingenstein. Federated security: The shibboleth approach. *EDUCAUSE Quarterly*, 27(4):12–17, 2004.

[30] NIST. INCITS 359-2004 - Role Based Access Control, 03 2004.

[31] OASIS. eXtensible Access Control Markup Language (XACML). https://www.oasis-open.org/committees/xacml (Last accessed May 2013).

[32] OASIS. XACML v3.0 Core and Hierarchical Role Based Access Control (RBAC) Profile, 2010.

[33] OMG. Meta Object Facility (MOF) 2.0.

[34] M. Rohr, M. Boskovic, S. Giesecke, and W. Hasselbring. Model-driven development of selfmanaging software systems. In *ACM/IEEE MoDELS Workshop on Models@Runtime*, 2006.

[35] R. Sandhu. The authorization leap from rights to attributes: maturation or chaos? In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*, SACMAT '12, pages 69–70. ACM, 2012.

[36] R. Sandhu, D. Ferraiolo, and R. Kuhn. The NIST model for role-based access control: towards a unified standard. In *Proceedings of the fifth ACM workshop on Role-based access control*, RBAC '00, pages 47–63. ACM, 2000.

[37] K. Sohr, G.-J. Ahn, and L. Migge. Articulating and enforcing authorisation policies with UML and OCL. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–7, 2005.

[38] E. Song, R. Reddy, R. France, I. Ray, G. Georg, and R. Alexander. Verifiable composition of access control and application features. In *SACMAT '05: Proceedings of the tenth ACM symposium on Access control models and technologies*, pages 120–129. ACM, 2005.

[39] W. Sun, R. France, and I. Ray. Rigorous Analysis of UML Access Control Policy Models. In *Proceedings of the 2011 IEEE International Symposium on Policies for Distributed Systems and Networks*, POLICY '11, pages 9–16. IEEE Computer Society, 2011.

[40] L. Wang, D. Wijesekera, and S. Jajodia. A logic-based framework for attribute based access control. In *Proceedings of the 2004 ACM workshop on Formal methods in security engineering*, FMSE '04, pages 45–55. ACM, 2004.