Model Driven Configuration of Secure Operating Systems for Mobile Applications in Healthcare

B. Agreiter[1], M. Alam[1], M. Hafner[1], J.-P. Seifert[14], and X. Zhang[4]

[1] University of Innsbruck, Austria
{berthold.agreiter, masoom.alam, m.hafner,
jean-pierre.seifert}@uibk.ac.at
[4] Samsung Information Systems America, San Jose, CA,USA
xinwen.z@samsung.com

Abstract. Trust and assurance of mobile platforms is a prime objective when considering their deployment to security-critical scenarios in e.g., healthcare or e-government. Currently, several complementary approaches are being pursued in parallel, ranging from purely hardware based, to operating system level, and application level solutions. Together, they build a "trusted and secured" technology stack. However, the very complex policy configuration mechanisms at every single layer also represent the biggest stumbling block for a rapid adoption. We propose a practicable and efficient solution for leveraging operating system level and application level security mechanisms to realize security-critical application and services for healthcare scenarios.

1 Introduction

In recent years the IT-industry experienced a significant increase in computing power, connection bandwidth, and data storage on mobile devices (e.g., cellular phones, smart phones, PDA). As a result, mobile devices are becoming general-purpose computing environments with the following main characteristics:

1. *New Business Scenarios.* Advanced applications and services scenarios are being deployed to these devices. They bring about new usage models and business processes (e.g., mobile payment, software as a service (SaaS), pervasive information and content (audio, video, and text) sharing, etc.).
2. *Multilateral Security.* Sensitive data are stored on a device, not only the user's sensitive information (e.g., credit card number and e-tickets), and network operator's sensitive data (e.g., subscriber identity and billing information), but also other services' data (e.g., protected content and DRM applications licenses), and data for platform management agents (e.g., anti-virus applications). This creates the requirement to support the trust needs of all these different stakeholders.
3. *Pervasiveness.* With increasing connecting capabilities such as voice and digital data from network service providers, WiFi/WLAN, Bluetooth, and UPnP, mobile and home computing has become more pervasive and ad-hoc than ever.

Trust and assurance of the computing environment on devices has to be enhanced when considering their use in security-critical scenarios (e.g., healthcare, e-government). Currently, several approaches are being pursued in parallel, ranging from purely hardware based approaches, to operating system based approaches to application-level security frameworks. Nevertheless, it is very important to keep in mind, that these initiatives – all focusing on different aspects of system security at various layers of the system stack – actually complement each other to build what is called a Trusted Platform [1].

A fundamental but purely hardware-based approach is led by the Trusted Computing Group (TCG) [1]. TCG has defined the specification of a trusted platform module (TPM) for PC-platforms, and the corresponding mobile phone working group in TCG is developing the specification of a mobile trusted module (MTM), which is to provide high assurance of the integrity of mobile platforms.

Going up one layer in the platform hierarchy, the next logical step is to tackle security at the level of the operating system. One of the most promising initiatives in that are is Security Enhanced Linux (SELinux). [13]. SELinux realizes the principle of least privilege by enforcing *Mandatory Access Control* (MAC) [11] based on *Type Enforcement*.

At the top most layer – the application layer – we find application-level security frameworks, dealing with the enforcement of security in context of application-, service-, and business-functionality. In the past years, much effort has been put into the realization of secure systems based on the paradigm of service oriented architectures and Web services (e.g., [4,5,26,6]).

Problem Statement. Unfortunately, the very complex policy configuration mechanisms at every single layer could potentially also represent the biggest stumbling block for a rapid adoption of the "trusted and secured" technology stack. Security experts keep on arguing that these technologies are far too complex to configure, leaving these systems still vulnerable to attacks. All the more, the inter-dependency of these complementing technologies certainly does not alleviate the burden of a correct set-up.

Contribution. Taking the complexity of these security-mechanisms and their dependencies as a starting point, we propose a practicable and efficient solution how these technologies may be leveraged to realize security-critical application and services for healthcare scenarios. As mobile trusted modules are not yet broadly available on the market and to remain independent from underlying hardware, we base our work on a hardened Linux (SELinux). The model-driven approach takes application-level policy models as input and generates configuration files for the operating system and application-level security mechanisms.

Organization. Section 2 motivates our research and gives a brief overview. Section 3 introduces our modelling approach guided by the principles of model driven security. Section 4 presents key technologies of the target architecture. Section 5 represents the core of this paper and describes policy generation. Section 6 puts our work in context of related work, and Section 7 gives an outlook on our research agenda.

2 System Overview and Motivation

Several approaches discuss usage control in distributed environments (cf. [21,20]). In this paper, we are focussing on usage control requirements especially in the context of healthcare and mobile devices.

2.1 Scenario

Security is a crucial issue in IT-supported healthcare services. We sketch an example scenario where data accessed through a mobile device should be protected.

Physicians use to carry their mobile devices (e.g. PDA or Smartphone) with them during work in a hospital. Due to alternating connectivity patient data is stored on these devices until synchronized with the hospital database. However, this also imposes several risks like e.g., the compromise of the patient records stored on the device when the device is taken out of the hospital.

A very basic threat may stem from the user of the device itself. Considering privacy regulations, the patient record should only be used within a certain scope.

This scenario indicates location-aware document access and implies role-based access control. According to the latter, physicians and nurses would not be allowed to execute the same operations on a given document.

2.2 Overview

To meet these security requirements we basically need an enforcing security architecture on devices accessing sensitive data. The security issues in this case are twofold:

1. Provide protection of the sensitive data (i.e. patient records, security configuration) on the device against potential malicious users or malware – thus enforcing multi-lateral security.
2. Provide fine-grained protection to the patient records to ensure only authorized personnel is granted the corresponding access to specific parts of the patient record – thus enforcing the principle of least privilege.

We solve the first issue by leveraging SELinux using MAC policies on the mobile device. This ensures that security configurations and patient records are protected against unauthorized access. The basic protection we achieve with this technique is file-level access and protection of the platform and its configuration, needed for successfully enforcing the fine-grained protection to the patient records. Clearly, some performance overhead is produced, but due to previous research and increasing computing power this is not considered a problem for now (cf. [25]).

In order to provide fine-grained access control to patient records, we rely on the UCON-security model [27]. UCON integrates authorizations, obligations and conditions and supports two concepts called *attribute mutability* and *decision*

continuity. The basic idea of these concepts is that access is not granted once and forever, but is checked throughout the session and can also be revoked.

Our architecture on the mobile device enforces these requirements. The usage policies are not pre-installed on the device itself, but are received from the document provider (i.e. the hospital service) in conjunction with the patient record. This allows the document provider to enforce its security and privacy rules also on the mobile devices.

3 Policy Modelling

Policies represent the central concept for realizing security within our work. We follow a model-driven approach to build these policies. In the following, we give example rules (aka "security requirements") that may be enforced through a respective policy, and we then present a domain specific language that can be used to formalize the policies.

3.1 Security Requirements & Models

A domain specific language (DSL) supporting the modeling of security requirements for healthcare information systems would certainly have to cater for a broad variety of domain specific security requirements. Henceforth, we solely focus on access control, coming in two flavours (Role-based Access Control and Mandatory Access Control) and usage control. For comprehensive discussion of healthcare specific security requirements, we refer to [3].

Access control is primarily concerned with authorization decisions involving a subject requesting some kind of access to a resource, possibly even on the grounds of currently available information and contextual constraints (dynamic access control).

Role-based Access Control (RBAC) extends the flexibilty of basic access models by introducing roles. We will use RBAC for the application level as it seems the most natural peer in terms of its semantics to application concepts and processes in healthcare. RBAC policies would state rules like:

"A user holding role `Physician` *can write into records of type* `Diagnosis`*."*

Contextual aspects could be added – making it a conditional RBAC based policy – and stated as a rule like, e.g.:

"A user holding role `Nurse` *can read records of type* `Diagnosis`*,*
on working days, between 9.00 and 17.00."

Mandatory Access Control (MAC). In systems based on MAC policies are also adminstered by a central authority, but system-wide rules can be specified which must be followed by everyone. One mechanism implementing MAC is *type enforcement*. It assigns *Security labels* to data elements to express their security sensitivity. Access decisions are then only based on these labels. We will integrate SELinux's type enforcement features to complement application level security based on RBAC. SELinux stores the labels directly with every labeled object (e.g. inode of a file).

To allow specific processes access to patient records, a system wide administrator would define a rule for a client-side operating system:

"Only allow the client-side `security_server`*-process read and write access to the file-system holding files of type* `HealthRecord`*."*

Usage control (UC) extends the concept of access control to what may and may not happen to a data item in the future, *after it has been given from a data provider to a data consumer*. A detailed discussion of the general class of usage control requirements encompassing data protection as well as management of intellectual property including a formal specification language can be found in [7]. We base our DSL on **UCON**, which is a comprehensive usage control model that extends the traditional access control models in two respects [20]:

1. continuity of access decisions: decisions to access an object are not only verified before but also during access and may result in a revocation of access permissions, if policy conditions are not satisfied.
2. mutability of attributes: this means that attributes of subject or object may change as side effects of an access, which may also result in a change of ongoing or subsequent access decisions.

Policy statements in UCON consist of *authorizations*, *obligations* and *conditions*. Authorizations refer to predicates which are based on subject or object attributes only. Obligation actions are directives to a subject to perform additional actions before or during an access. Predicates exclusively based on environment attributes such as system time, device type etc., are categorized as conditions. *A*uthorizations, o*B*ligations and *C*onditions are collectively referred to as UCON_{ABC}. An example rule would be stated like e.g.,:

"A user holding role `HealthPersonnel` *can read any* `HealthRecord` *but is not allowed to further disseminate the* `Document`*."*

For an in-depth coverage on the model driven realization of healthcare specific security requirements, we refer to [3,5,4,14,8].

3.2 Domain Specific Language

The modeling of security aspects for healthcare scenarios occurs through *Healthcare Policy Models*, based on an extended SECTET-DSL [18,9,16]. The purpose of the DSL is to integrate and visually render UCON-concepts the context of sub-model elements from SECTET (e.g., Document-, Role-, Access-Model).

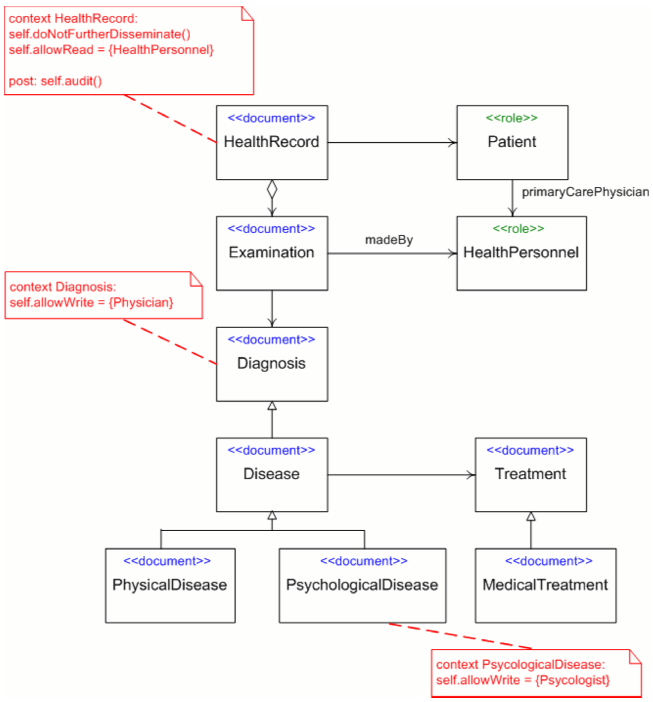

Fig. 1. Sample Healthcare Policy

Policy Model. Figure 1 shows an example healthcare policy incorporating some of the rules stated in Section 3.1. UCON subjects are modelled as UML classes stereotyped with `<<role>>` and UCON objects are modelled as UML classes stereotyped with `<<document>>`. UML stereotypes specify the sub-models at the metamodel level. Security requirements are defined by attaching constraint boxes to the classes. Details on syntax and semantics of the UML-profile can be found in [8,2,15].

Meta-Model. The metamodel defines the domain specific language for our security-critical scenarios in healthcare. Elements reference elements of other (sub-)models through proxy classes, as for example `RoleRef` references a `Role` from the Role Metamodel.

The pattern `UsageControl` is defined as a `DomainSpecificSecurity-Requirement`. It is associated to the UCON element `Right` and is conditionally associated with an `Authorization`, an `Obligation`, and/or a `Condition`. In our case, `Authorization`, `Obligation`, and `Condition` are specified using SECTET-PL [15].

Policy Generation. Taking these policy models as input, the SECTET-Framework [18,9] generates two policy files, which configure two sub-components of the target reference archtitecture: (1) an XACML Usage Policy, defining usage

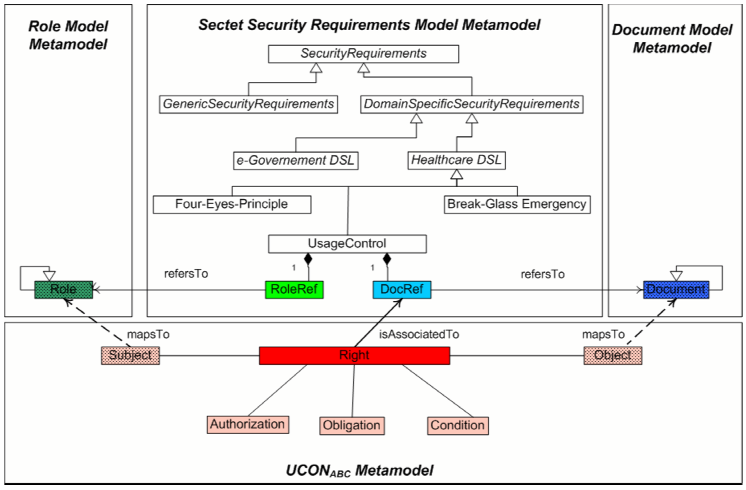

Fig. 2. SECTET–Domain Specific Language

and security policies for user space application and security components (e.g., Fig. 3), configures the policy server; (2) an SELinux Kernel Policy, defining the operating system-level security policies (please, refer to Sec. 4.2).

Subsequently, we will briefly sketch structure, syntax and semantics of an sample usage policy, kernel policies are described in the next section.

The XACML policy file configures a user space security server (cf. Fig. 4). Policies are either stored locally upfront at the client's site or may be sent along with the document. This *usage policy* is generated from the policy model or, more precisely, from the constraints attached to a usage control restricted resource. Referring to our examples in Section 3.1, Fig. 3 shows how access to and usage control requirements for the resource `HealthRecord` may be transformed.

The policy generated for the resource `HealthRecord` and role `HealthPersonnel` contains the authorization and obligation constraints in the form of `<Rule>`s. The `<Target>` element contains the name of the document (line 08) and of the operation (line 11) to which the `<Rule>` applies. A `<Condition>` element additionally specifies an authorization constraint. If the authorization constraint is met by the subject, access will be granted. The `<Condition>` element defines authorization constraints in the form of XACML functions.

The right-hand side of Figure 3 partially describes an example `Obligation Constraint`. According to the usage control requirement stated in Section 3.1, the obligation, *"Users assigned roles* `HealthPersonnel` *should not be allowed to disseminate documents of type* `HealthRecord`*. Every access to the ressource should be logged."*

Extended obligation functions like `SECTET:obligation:function: cannotSpreadFurther` and `SECTET:obligation:function:post_audit` obligates the *policy server* to prohibit further dissemination and to log every access.

Fig. 3. Example Generated User Space Policy (simplified XACML syntax)

4 Policy Enforcement

To enforce the proposed security requirements we leverage security functionality from the operating system as much as possible. This keeps the security architecture on the lower layers of the software stack, which is again beneficial for reducing the complexity of the attestation, which means, deciding whether a device is trustworthy or not. Our choice for this architecture leverages SELinux. It enforces MAC policies and protects data from any illegal access.

4.1 Architecture

As already stated, there are two distinct levels of security requirements to be enforced by the system. The fine-grained usage control decisions are made and enforced by a *userspace security server*. Every action to be taken on a document has to be allowed by it. Each document is *only* accessible through the userspace security server and moreover, it is the only application on the system allowed to access patient records. Any application other than the security server will be denied access. How the access-denial from other components is realized will be explained in a later section.

The security server is a component working according to the UCON model. This means it may update attributes about the subject and object when objects are accessed. The policies enforcing the condition checks and attribute updates are XACML policies described in section 3, we refer to them as *usage policies*. The security server has an attribute repository where it stores the user and object attributes for each document.

Upon access to a document, the viewer application submits the intended action to the security server, which fetches the corresponding usage policy from the Policy Management Server. The security server updates attributes and checks

conditions according the policy and if access is granted, the desired actions are taken on the document (e.g. add a new prescription). The advantage of this architecture is the domain-specific knowledge of the security server. Due to this, it can enforce security requirements on an arbitrarily fine level on the documents. As an example, only physicians should be allowed to add prescriptions to a patient record, for any other role these fields should be read-only or not visible at all. Nevertheless, we claim the security server is generic enough to be configured for other domains than healthcare.

The *Gateway* constitutes the interface to the document providers. It is also restricted in its capabilities by the kernel policy. When the Gateway receives a document it separates the data from the policy. The data is stored in the filesystem and the policy is passed to the Policy Management Server. After that, the document is ready for use.

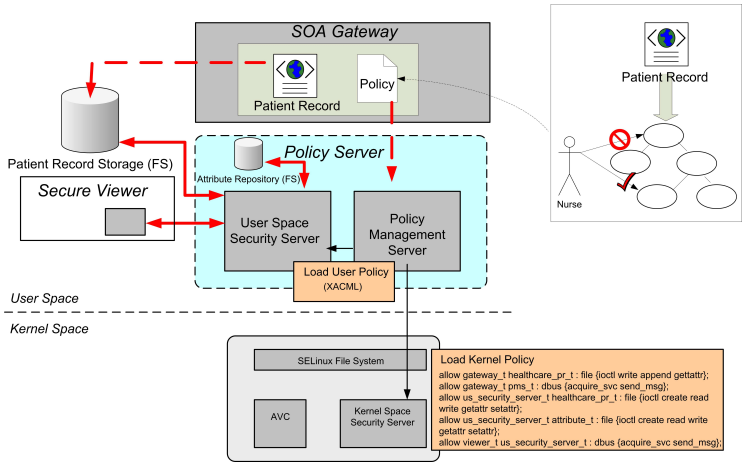

Fig. 4. Architecture

4.2 Kernel Policy

To make sure a document can not be compromised when it is stored on a device an additional security mechanism is needed. Otherwise, documents would at least be accessible to an administrator of the system, which is able to override the patient record's user policies. To fulfill this need we are using SELinux, which provides us with the ability to specify access rules to fine-grained operating system objects (e.g. files, network sockets, IPC, etc.).

Specifically, we only allow certain applications access to sensitive data. Figure 4 shows which component is allowed access on which other component or the filesystem (bold arrows). To provide an example, only the security server is

allowed to access the patient records stored on the file system. Any other process will be denied access to this data.

An access decision is based on labels, called security context in SELinux. Every subject and object is associated a label of the form `(user:role:type)`. For subjects the user and role are determined by the current user. The type is the most important field and may be different depending on which application a process is currently executing. Upon accessing any object, the kernel's security server (cf. 4) is asked for permission. This decision is based on the kernel policy.

Every application is assigned a security context upon installation on the device. Furthermore, when patient records are received and stored on the device they are automatically labeled. Different techniques can be applied to achieve this: either the Gateway uses SELinux API-calls to assign the security context, or it is inherited from a parent (e.g. parent directory where the file is saved).

The kernel policy holds the configuration for the components and hence, enforces the coarse-grained access. As already mentioned, the security server is the only component allowed to access patient records. To enforce this behavior the patient records are labeled `(user_u:object_r:healtcare_pr_t)` and the process executing the security server is labeled `(user_u:system_r:us_security_server_t)`. To only allow this process to access patient records we define a rule in the kernel policy:

```
allow us_security_server_t healthcare_pr_t: file {ioctl create \
read write append getattr setattr };
```

Similar rules are also defined to access the attribute repository and the communication between the viewer application and the security server. Note that there is no notion of a superuser able to override the rules defined in the policy.

5 Policy Generation & Model Transformation

Model-driven aspects are the second building block of ongoing work. In this context, the generation of the usage control policy is an ideal candidate. The document provider creates a policy by modeling the document model as a UML class diagram and annotates it with usage relevant information. The configuration is then based on two (or more) sub-models: the document model containing usage- and meta-information about patient records, and the role model defining the role hierarchy in the usage context. The mapping and transformation between document model and usage policies is outside the scope of this contribution, but is subject of ongoing research [3].

Another candidate for semi-/automatic generation is the kernel policy. It restricts the capabilities of each component running on the target device and provides the necessary groundwork to protect sensitive data at the operating system level. The kernel policy is dependent on two aspects:

1. *The architectural blueprint.* The different components in user space are assigned different types and they communicate with each other in various different ways (e.g. TCP/IP, D-BUS).

2. *The application artifacts.* Depending on the data which is received by application components, different rules need to be enforced (i.e. data file vs. executable file).

5.1 Pattern Discovery

SELinux Patterns. For both of these aspects we plan to use a similar approach. The ultimate goal is basically to identify syntactical patterns (or "macros") for the kernel policy that correspond to various categories of domain-level security concerns. We tackle this by visualizing the circumstances under which components could do their work properly if only given minimal access to certain objects. In a first step, this technique is helpful for finding patterns for different usage and security scenarios. In a second step, we link these patterns to a more abstract, semantically enriched representation, e.g., a security server may be the only component allowed to access patient records. The abstract information should finally serve as input for the generation of a platform's kernel policy.

Pattern Discovery. A possible technique for identifying kernel policy patterns is the following: in a simulation run, we first allow all subjects access to all objects (SELinux `permissive` policy mode). Note, that the built-in DAC access control mechanism of the operating system must be configured to allow every access, since it is evaluated before the SELinux mechanism is queried. This guarantees that for every usage of objects the SELinux *Access Vector Cache* is queried and logged, but its decision is not enforced. Instead, only audit logs are generated. These logs serve as input for the next step, which is the identification of patterns (analysis phase). Tools like `seaudit` are used to analyze the audit log created during the simulation phase. The policy is developed iteratively and the same steps are repeated for different usage scenarios.

This procedure is used for discovering patterns in the kernel policy. These patterns are used to identify types for subjects and objects and their usage in rules. This procedure of observing runtime behavior and finding policy types and rules most likely be supported by policy generation tools like Polgen or Madison ([17], [23]).

5.2 Policy Generation

As already stated, we use these generated policies to identify patterns in them. These patterns are stored in a *pattern repository* and can be exported if comparable security requirements have to be enforced in a system. Patterns in the *pattern repository* may be very specific to certain architectures or application domains.

The policy generation procedure is shown in Fig. 5. When a policy for a specific platform has to be created, the generator is fed with the kind of application artifacts to deal with (e.g. data files, executables) and the architectural blueprint

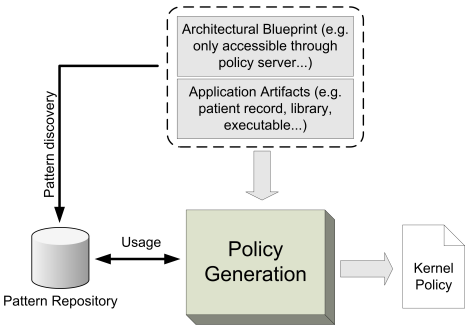

Fig. 5. Policy generation and usage of patterns

(e.g. which components exist, how do they communicate). Based on this information, it searches the pattern repository for usable patterns and imports the matching ones into the kernel policy. If the policy generation mechanism is given an input where no patterns are available in the repository another pattern discovery process has to be made for the current input.

Note that there is still need for validation in two places: (1) before patterns are put into the repository and (2) after their combination. We leave this issue open for future investigation.

6 Related Work & Discussion

6.1 (Model Driven) Security Engineering

We identified the following areas of work related to Model Driven (Security) Engineering.

In [22] the author presents an engineering approach based on pattern-based software development. The basic idea is to capture expert-knowledge in security and make it available to developers as patterns during software development. The approach gives a comprehensive introduction to security patterns, the pattern-based development-process, and describes the relationship between various security patterns. Although the author uses patterns to systematically capture knowledge about security issues at the model level, the semantics remain close to the technical level. The author does not address transformation between different layers of abstraction. Our approach raises the level of abstraction and also provides a methodology to systematically transform abstracts models into runtime artefacts. A very intersting approach for building secure systems is presented in [19]. It is based on an agent-oriented approach using security patterns. However, the authors only focus on the modelling aspects of secure systems engineering and seek to prove completeness of the patterns by formalizing their properties. The approach does not consider any code generation.

In [24] and [12], the authors introduce the concept of Model Driven Security for a software development process that supports the integration of security re-

quirements into system models. The models form the input for the generation of security infrastructures. However, the approach focuses exclusively on access control in the context of application logic and targets object oriented platforms (.Net and J2EE). Our approach differs in many ways. First, we consider various categories of security requirements, access control being alone one of them. Second, we raise the level of abstraction and consider security from the perspective of domain experts. And third, we target various layers of the software stack, the runtime being one of them.

[10] presents a verification framework for UML models enriched with security properties through a UML profile called UMLSec. The framework stores models using XMI format in a Meta Data Repository, which is then queried using Java Metadata Interfaces by different analyzers. These analyzers perform static as well dynamic analysis on the UMLSec models for security properties like confidentiality and integrity. This approach is orthogonal to ours and the abstraction is close to the technical level, whereas our framework is domain specific and focuses on the systematic generation of (standard) security artefacts specified during the early phases of software development. Our objective is to develop abstract languages for the realization of security requirements in distributed systems Tools and Frameworks.

6.2 SELinux Policy Engineering

A related approach for the user-guided and semi-automated policy generation of SELinux policies can be found in [23] and [17]. The two approaches are comparable and have similar ambitions. The Polgen-tool processes traces of the dynamic behavior of a target program and observes information flow patterns. It has a pattern-recognition module and creates new types according the patterns it detects. The policy generation process is interactive and human-guided. In contrast to Polgen and Madison we do not aim at creating SELinux policies by observing dynamic application behavior. Our efforts are towards building a policy generation framework for a specific target architecture and a specific application based on abstract domain-level securtiy concerns. We basically do this by using policy patterns from a repository, but for filling the repository the tools mentioned could be of great use. Another tool which is similar to the ones mentioned is audit2allow.

We are currently investigating alternative ways on how to use these tools for our work, especially the pattern discovery phase for building the pattern repository.

7 Conclusion and Outlook

This contribution presented an architecture and an approach for model-driven configuration of usage control requirements specific to healthcare. The security configuration is split up into fine-grained rules for the enforcement of document-

and domain-specific requirements, and OS-level security configuration for protection of running components and coarse-grained access-control. Furthermore, we presented our idea of a pattern repository which is consulted when creating new policies. Policies are generated by presenting the architectural blueprint and the application artifacts the system is working with. A more detailed description of the policy generation is left for future work.

Our architecture leverages the low-level security enforcement architecture SELinux which can be easily integrated in the Linux kernel since version 2.6. We are using it for a specific target architecture instead of protection from possible software failures. We claim it is more suitable in respect to performance for mobile devices compared to encryption of patient records and virtualisation for building an isolated sandbox, but leave proof to this claim open for future work.

Another interesting topic left for future work is the validation of policy patterns and its combination into a complete policy or policy module, respectively. Before patterns are put into the pattern repository they should be proven to really enforce the requirements they claim to, otherwise such a pattern could invalidate the whole repository.

References

1. Trusted computing group (tcg). https://www.trustedcomputinggroup.org/specs/.
2. M. Alam, M. Hafner, Ruth Breu, and Steffan Untertheiner. A Framework for Modelling Restricted Delegation of Rights in the SECTET. International Journal of Computer Systems, Science and Engineering devoted to Best TrustBus 06 Conference Papers.
3. M. M. Alam, M. Hafner, M.M. Memon, and P. Hung. Modelling and enforcing advanced access control policies in healthcare with SECTET. *Submitted.*
4. Masoom Alam, Michael Hafner, and Ruth Breu. Modeling authorization in an soa based application scenario. In *IASTED Conf. on Software Engineering*, pages 79–84, 2006.
5. Masoom Alam, Michael Hafner, Ruth Breu, and Stefan Unterthiner. A framework for modeling restricted delegation in service oriented architecture. In *TrustBus*, pages 142–151, 2006.
6. R. Bhatti, E. Bertino, and A. Ghafoor. Access control in dynamic xml-based web-services with x-rbac. ICWS 2003, LNCS 2722.
7. M. Hilty et al. Enforcement for Usage Control: A System Model and a Policy Language for Distributed Usage Control. Technical Report I-ST-20, DoCoMo EuroLabs, 2006.
8. M. Hafner, B. Agreiter, R. Breu, and A. Nowak. Sectet an extensible framework for the realization of secure inter-organizational workflows. *Journal of Internet Research*, 16(5), 2006.
9. M. Hafner, B. Agreiter, R. Breu, and A. Nowak. Sectet an extensible framework for the realization of secure inter-organizational workflows. *Journal of Internet Research*, 16(5), 2006.
10. Jan Juerjens. *Secure Systems Development with UML.* Springer Academic Publishers, 2004.

11. Donald C. Latham. *Department of Defense Trusted Computer System Evaluation Criteria*. DEPARTMENT OF DEFENSE, http://csrc.nist.gov/secpubs/rainbow/std001.txt, 1986.
12. Torsten Lodderstedt, David Basin, and Jürgen Doser. SecureUML: A UML-based Modeling Language for Model-Driven Security. In *Proceedings LNCS 2460*, pages 426–441. Springer, 2002.
13. Peter Loscocco and Stephen Smalley. Integrating flexible support for security policies into the linux operating system. In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, pages 29–42, Berkeley, CA, USA, 2001. USENIX Association.
14. M. Alam and M. Hafner and R. Breu. A Constraint based Role Based Access Control in the SECTET A Model-Driven Approach. International Journal of Information Security.
15. M. Alam et al. Modeling Permissions in a (U/X)ML World. In *IEEE ARES*, 2006.
16. R. Breu M. Hafner, M. Alam. A mof/qvt-based domain architecture for model driven security. In *IEEE/ACM Models 2006 LNCS 4199*.
17. Karl MacMillan. Madison: A new approach to automated policy generation. March 2007.
18. Hafner Michael. `SECTET` A Domain Architecture for Model Driven Security. PhD Thesis November 2006.
19. H. Mouratidis, M. Weiss, and P. Giogini. Modelling secure systems using an agent-oriented approach and security patterns. *International Journal of Software Engineering and Knowledge Engineering*, 2005.
20. J. Park and R. Sandhu. The UCON ABC Usage Control Model. *ACM Transactions on Information and Systems Security*, 7:128–174, 2004.
21. Alexander Pretschner, Manuel Hilty, and David Basin. Distributed usage control. *Commun. ACM*, 49(9):39–44, 2006.
22. Markus Schumacher. *Security Engineering with Patterns. Origins, Theoretical Models, and New Applications*. Springer, 2003.
23. Brian T. Sniffen, David R. Harris, and John D. Ramsdell. Guided policy generation for application authors. February 2006.
24. T. Lodderstedt, D. Basin and J. Doser. A UML Based Modeling Language for Model-Driven Security . 5th international conference UML 2002 Dresden, Germany, 2002.
25. C. Wright, C. Cowan, J. Morris, S. Smalley, and G. Kroah-Hartman. Linux security modules: general security support for the linux kernel. *Foundations of Intrusion Tolerant Systems, 2003 [Organically Assured and Survivable Information Systems]*, pages 213–226, 2003.
26. E. Yuan and J. Tong. Attributed Based Access Control (ABAC) for Web Services. IEEE ICWS 2005,ISBN 0-7695-2409-5.
27. Xinwen Zhang, Francesco Parisi-Presicce, Ravi Sandhu, and Jaehong Park. Formal model and policy specification of usage control. *ACM Trans. Inf. Syst. Secur.*, 8(4):351–387, 2005.