



HAL
open science

Automatic reconstruction and analysis of security policies from deployed security components

Salvador Martínez

► **To cite this version:**

Salvador Martínez. Automatic reconstruction and analysis of security policies from deployed security components. Systems and Control [cs.SY]. Ecole des Mines de Nantes, 2014. English. NNT : 2014EMNA0193 . tel-01065944

HAL Id: tel-01065944

<https://theses.hal.science/tel-01065944v1>

Submitted on 18 Sep 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Thèse de Doctorat

Salvador MARTÍNEZ

*Mémoire présenté en vue de l'obtention du
grade de Docteur de l'École nationale supérieure des mines de Nantes
sous le label de l'Université de Nantes Angers Le Mans*

Discipline : Informatique et applications

Laboratoire : Laboratoire d'informatique de Nantes-Atlantique (LINA)

Soutenue le 30 Juin 2014

École doctorale : 503 (STIM)

Thèse n° : 2014 EMNA 0193

Automatic Reconstruction and Analysis of Security Policies from Deployed Security Components

JURY

- Rapporteurs : **M. Jacky AKOKA**, Professeur, Conservatoire National des Arts et Métiers
M. Régine LALEAU, Professeur, Université Paris-Est Créteil
- Examineur : **M. Christian ATTIOGBÉ**, Professeur, Université de Nantes
- Directeur de thèse : **M. Frédéric CUPPENS**, Professeur, Télécom Bretagne, Campus de Rennes
- Co-directeur de thèse : **M. Jordi CABOT**, Maître de conférences, HDR, École des Mines de Nantes

Acknowledgements

However being an individual challenge, the works of this Ph.D thesis would have not been accomplished without the help and care of many colleagues and friends. The following lines are devoted to thank all those who helped me.

First of all, I would like to thank my thesis advisor, Jordi Cabot. He fought to get this thesis started and, during all the time it lasted, he provided me with a perfect equilibrium between the freedom to explore my own ideas and good advise and direction.

The works that lead to the defense of this thesis have been performed in the framework of a very good research team, AtlanMod. Among its members I have found not only enriching colleagues, but also friends. In that sense, I would like to thank Fabian Büttner, Javier Cánovas and Valerio Cosentino. From discussing about modeling, to drinking beers passing by killing zombies, they were a support for making the time I have spent working in my thesis not only fruitful and enriching from the professional side, but also very enjoyable from the personal one.

Finally, I would like to specially mention Sirle Trestip and Anne Bigot with whom I have shared uncountable evenings, challenges and bets that helped me to overcome the hard times of writing a Ph.D thesis.

Contents

1	Resumé	7
1.1	Introduction	7
1.2	Description du problème	9
1.3	Framework dirigé par les modèles	10
2	Introduction	15
2.1	Problem description	16
2.2	Basic Concepts	22
2.3	State of the art	30
2.4	Thesis Structure	36
3	Objectives and Contributions	39
4	Model-driven Framework	43
4.1	Platform specific models (PSMs)	43
4.2	Extracting abstract (PIM) models	45
4.3	CIM	45
5	Extracting AC Models from Networks	49
5.1	Motivation	51
5.2	Approach	53
5.3	Applications	64
5.4	Analysis of stateful misconfigurations	66
5.5	Tool Support	73
6	Extracting AC Models from RDBMS	77
6.1	Motivation	78
6.2	RDBMS Access-Control Metamodel	81
6.3	Reverse Engineering Process	84
6.4	Applications	87
6.5	Tool Support	92

7	Extracting AC Models from CMSs	95
7.1	Motivation	96
7.2	WCMS Access-Control Metamodel	97
7.3	Approach	101
7.4	Applications	103
8	Integration	107
8.1	AC Integration for multi-layer systems	107
8.2	Motivation	108
8.3	Approach	110
8.4	Policy Translation	111
8.5	Integration	115
8.6	Policy Analysis Support	118
8.7	Tool support	123
9	Related Work	127
9.1	Network AC Reverse-engineering	127
9.2	RDBMS AC Reverse-engineering	132
9.3	CMS AC Reverse-engineering	134
9.4	AC Policy Integration	135
10	Conclusion	137
10.1	Conclusion	137
10.2	Future Work	139
	Publications	143

Resumé

1.1 Introduction

Les systèmes d'information de l'entreprise sont, le plus souvent, composés d'un nombre de composants hétérogènes, responsables de l'hébergement, la création ou la manipulation d'information critique pour l'opération de l'entreprise. La sécurisation de ces informations est donc une préoccupation essentielle. En ce sens, les propriétés de disponibilité, confidentialité ou intégrité (parmi beaucoup d'autres propriétés de sécurité), peuvent être des exigences pour le bon fonctionnement des données gérées par un système d'information.

Dans le but d'assurer ces propriétés de sécurité, chercheurs et fournisseurs d'outils ont proposé et développé au cours des dernières décennies une pléthore de mécanismes de sécurité visant à assurer que les données d'un système d'information (et le système lui-même) sont à l'abri des menaces possibles. En outre, bon nombre de ces mécanismes ont été intégrés dans les composants concrets des systèmes d'information (comme les bases de données, les systèmes d'exploitation, les serveurs web, etc) faisant de la sécurité un élément essentiel d'un grand nombre de ces composants.

Parmi toutes les propriétés de sécurité qu'un système d'information doit exiger, la confidentialité et l'intégrité, les propriétés de sécurité assurant que: 1) l'information est disponible seulement aux parties autorisées et 2) l'information est exacte et cohérente, sont deux des plus critiques. Un défaut de sécurisation de ces propriétés quand elles sont nécessaires, conduit à un compromis dans l'information. Ce compromis, peut causer d'importantes pertes à l'entreprise. En raison de sa relative simplicité conceptuelle, l'un des mécanismes les plus utilisés dans les com-

posants des systèmes d'information pour assurer la confidentialité des données et l'intégrité est la définition et la mise en œuvre des politiques de contrôle d'accès (AC). Grosso modo, les politiques de contrôle d'accès établissent pour un système donné un ensemble de règles indiquant quelles actions un acteur donné peut effectuer sur une ressource d'information.

Cependant, malgré les quelques méthodes qui tentent d'obtenir ces implémentations de la politique à partir des spécifications de sécurité de haut niveau, la mise en œuvre d'une politique de contrôle d'accès reste, dans la grande majorité de cas, complexe et source d'erreurs car elle impose la nécessité de connaître les outils et les techniques spécifiques de bas niveau et les détails des fournisseurs. En outre, dans le cas de systèmes complexes, composés d'un certain nombre de sous-systèmes hétérogènes en interaction, le contrôle d'accès est omniprésente par rapport à cette architecture. Nous pouvons trouver l'application de politiques de contrôle d'accès dans les différents composants placés à différents niveaux d'architecture. Par conséquent, dans n'importe quel système, un ensemble de politiques de contrôle d'accès met en œuvre les objectifs de sécurité. Mais ces politiques ne sont pas indépendantes et des relations existent entre elles, de la même façon que des relations existent entre les composants situés dans les différentes couches de l'architecture.

Étant donné que l'analyse de la politique de un composant dans l'isolement ne peut pas fournir suffisamment d'informations, idéalement, une représentation globale de la politique de contrôle d'accès de l'ensemble du système devrait être disponible. Toutefois, dans les faits, cette politique globale n'existe que d'une manière implicite et pas toujours cohérente.

Dans ce contexte, la découverte et la compréhension des politiques de sécurité qui sont effectivement appliquées par le système d'information apparaît comme une nécessité impérieuse. C'est une condition nécessaire pour la refonte des politiques actuelles que doivent s'adapter à l'évolution des besoins de l'entreprise et aussi pour détecter les incohérences entre les politiques appliquées et les politiques souhaitées. Le principal défi à résoudre est de combler l'écart entre les fonctions de sécurité dépendant des fournisseurs et une représentation de haut niveau qui exprime ces politiques d'une manière qui fait abstraction des spécificités des composants du système et qui puisse être compris par les experts en sécurité sans aucune connaissance approfondie des particularités de chaque technologie et fournisseur. Ce modèle logique nous permettrait également de mettre en œuvre toute opération d'évolution / refactoring / manipulation sur les politiques de sécurité, et ce d'une manière réutilisable.

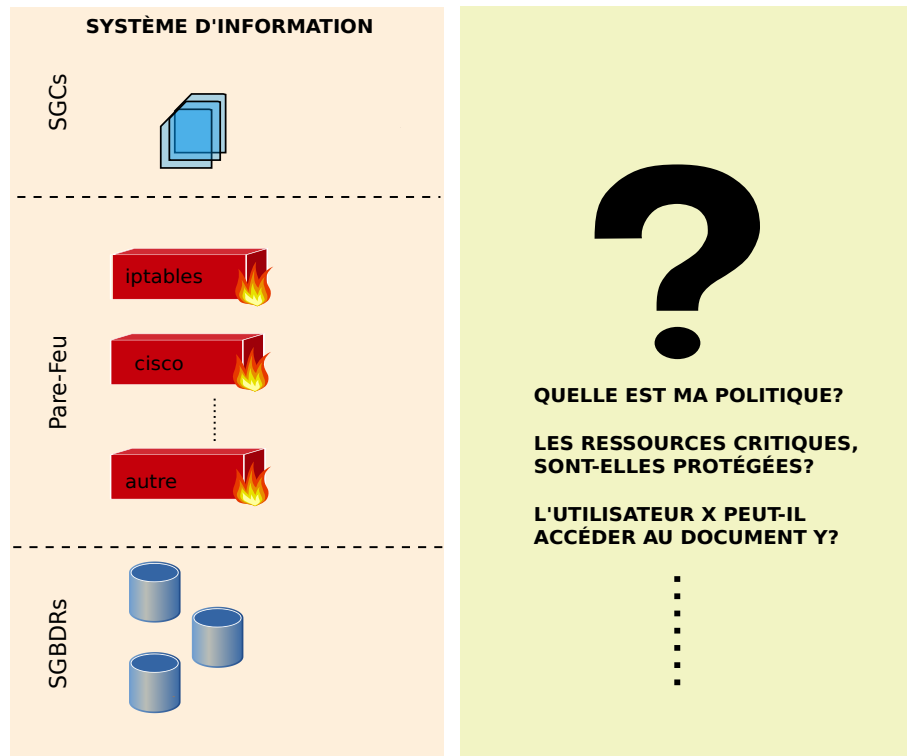


Figure 1.1: Description du problème

1.2 Description du problème

Comme présenté ci-dessus, la définition et mise en œuvre de politiques de contrôle d'accès est souvent le mécanisme de choix pour assurer la confidentialité des systèmes d'information. Cependant, la mise en œuvre de ces mécanismes reste complexe et sujette aux erreurs. Dans un système d'information complexe, avec plusieurs composants appliquant des politiques de contrôle d'accès, deux questions essentielles se posent:

1. Est-ce que la politique déployée dans un système concret a atteint l'objectif de sécurité? C'est à dire, est-ce que la politique déployée applique la confidentialité et l'intégrité comme souhaité?
2. Lorsque elles travaillent dans un système complexe dont elles font partie, les politiques des composants sont-elles compatibles les unes avec les autres de sorte que 1) L'objectif global de sécurité est atteint et 2) Le fonctionnement de chaque composant n'est pas touché de façon inattendue par les politiques des autres composants?

Malheureusement, répondre aux questions ci-dessus est une tâche très complexe. Cela est dû principalement à deux facteurs. 1) Le problème de la compréhension des politiques mises en œuvre dans des composants concrets 2) Le problème de l'analyse de toutes les politiques dans un système d'information dans son ensemble.

En ce qui concerne chaque composant concret, la complexité même des outils et des techniques de mise en œuvre a un impact important non seulement au moment de la mise en œuvre, mais aussi quand la politique doit être inspectée pour être analysée. Les configurations finales sont souvent représentées en utilisant des représentations de bas niveau, et spécifiques au fournisseur, comme des fichiers texte ou des tables de dictionnaire dans une base de données. En outre, plusieurs mécanismes peuvent participer à la mise en œuvre de la politique, ce que implique la repartition de la politique et, par conséquent, une complexité accrue.

Cependant, les politiques ne peuvent pas être considérées comme isolées, étant donné que des relations de dépendance existent entre les composants et leurs politiques. Malheureusement, ces relations entre les composants et leurs politiques de contrôle d'accès sont rarement explicites, ce qui les empêche d'être analysées dans leur ensemble. En supposant que le composant C_x dépend du composant C_y pour fonctionner correctement, le schéma d'autorisations de C_y , tandis que correct quand on le considère de manière isolée, peut mettre en danger le composant C_x ou l'empêcher de fonctionner correctement.

1.3 Framework dirigé par les modèles

Afin de résoudre le problème mentionné ci-dessus, et en raison de son efficacité avérée pour la spécification et la réalisation de systèmes complexes, nous proposons la construction d'un mécanisme automatique de rétro-ingénierie dirigée par les modèles. Ce mécanisme doit être capable d'analyser les aspects de sécurité des composants déployés (par exemple, configurations des pare-feu) pour dériver des modèles abstraits (Platform-independent model (PIM)) représentant la politique (par exemple, la politique de sécurité du réseau) qui est effectivement appliquée sur des composants du système. Une fois que ces modèles abstraits sont obtenus, ils peuvent être conciliés avec les directives de sécurité attendues, afin de vérifier leur conformité, ils peuvent être interrogés pour tester la cohérence ou utilisés pour générer des configurations de sécurité correctes. Enfin, ils peuvent être réunis pour représenter la politique de contrôle d'accès globale en utilisant un modèle des exigences (Computation Independent Model (CIM)) pour recueillir toutes les informations représentées dans les modèles PIM.

Dans le but de valider notre approche, nous avons choisi de l'appliquer sur trois systèmes différents et sur le système obtenu par leur combinaison. Concrètement, nous avons décidé d'appliquer notre approche sur: Les réseaux (concrètement, des pare-feu de réseaux), les systèmes de gestion de base de données (SGBDRs) et les systèmes de gestion de contenu (SGCs). Notons cependant que les résultats de cette thèse peuvent être étendus à d'autres sous-systèmes différents.

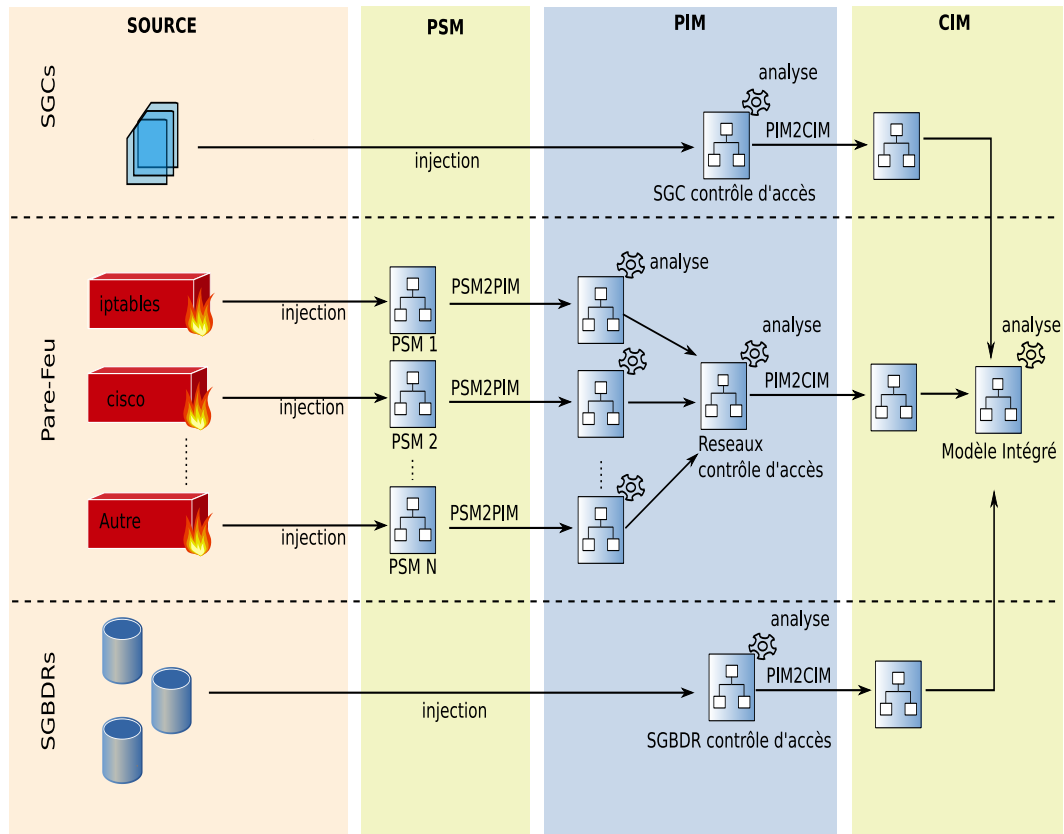


Figure 1.2: Vue d'ensemble du framework proposé

Ces trois systèmes travaillent dans différentes couches de l'architecture et sont souvent mis en relation pour construire des systèmes plus complexes. En outre, la mise en œuvre du contrôle d'accès est une caractéristique de base de chacun d'eux. Les pare-feu fournissent contrôle d'accès aux réseaux en filtrant le trafic en fonction des conditions données. SGBDRs et SGCs appliquent politiques de contrôle d'accès sur les données qu'ils stockent (et aussi sur son administration) en utilisant un certain nombre de règles de contrôle d'accès affirmant ce que les utilisateurs peuvent faire.

Dans la figure 1.2, nous décrivons la synthèse de l'approche proposée pour l'extraction et l'analyse des politiques de contrôle d'accès. Dans ce qui suit, nous allons décrire brièvement les étapes qui la composent.

1.3.1 Modèle technique: Platform specific models (PSMs)

La distance conceptuelle entre la source d'information et les modèles abstraits est généralement importante, ce qui rend le travail de l'injecteur (l'outil en charge de transformer les informations d'origine en un modèle) très complexe. Par conséquent, la définition des modèles techniques, à savoir les modèles représentant des informations contenues dans le même niveau d'abstraction que les informations de

source, peut être nécessaire. Ces modèles font comme un moyen de combler les espaces techniques, de sorte que nous passons de l'espace technique de la source d'information à l'espace technique ModelWare. Une fois dans ModelWare, les modèles techniques peuvent être transformés en modèles abstraits en utilisant des outils de transformations de modèles comme ATL [51].

A titre d'exemple, dans le cas des pare-feu, l'information de contrôle d'accès est stockée sous la forme de fichiers de configuration textuelles écrites à l'aide, normalement, de langages spécifiques au fournisseur avec différentes syntaxes et sémantiques. Afin d'extraire les informations de contrôle d'accès et de les représenter dans un modèle abstrait, analyseurs et injecteurs sont nécessaires. L'analyseur est capable de lire la source de l'information alors que l'injecteur utilise les informations obtenues afin de remplir un modèle avec ces informations. Étant donné que la variabilité et la distance sémantique des langages pare-feu par rapport au modèle abstrait sont grandes, analyseurs et injecteurs peuvent devenir trop compliqués. Ainsi, dans le but de simplifier la tâche, les informations sont d'abord extraites à un modèle technique, spécialement adapté pour la représentation de l'information AC d'un fournisseur particulier.

Au contraire, si nous jettons un coup d'oeil à l'information du contrôle d'accès dans les bases de données relationnelles, nous observons une situation contrastée. Pour l'extraction de l'information du AC stockée dans les tables du dictionnaire, un modèle PSM n'est pas nécessaire, puisque les concepts sont similaires à ceux du modèle abstraite. Toutefois, pour l'extraction des informations de contrôle d'accès contribué par les triggers et les procédures stockées, un PSM représentant le langage utilisé pour définir les procédures stockées est nécessaire car la distance entre le modèle abstrait et le code source est grande.

1.3.2 Modèle métier: Platform-independent models (PIM)

L'extraction de modèles PSM crée un pont entre l'espace technique de la source des informations de configuration et le ModelWare. Avoir les informations de contrôle d'accès d'un système représentées comme un modèle permet la réutilisation des outils de l'ingénierie dirigée par les modèles (IDM). Des générateurs de l'éditeur, des moteurs de transformation de modèle, etc., deviennent automatiquement disponibles.

Cependant, l'information des modèles de PSM est au même niveau d'abstraction que les informations de la configuration initiale. Ainsi, la étape suivante est d'élever le niveau d'abstraction de l'information contenue dans ces modèles. Cette information est alors représentée dans un modèle métier (PIM), de sorte qu'il puisse être manipulé sans tenir compte des spécificités des technologies utilisées pour sa mise en œuvre (remarquons que lorsque nous nous référons ici à un modèle métier, nous

avons l'intention de souligner que les modèles sont indépendant des spécificités des fournisseurs concrets, tout en étant spécifiques au domaine du composant). Au coeur de cette étape est le développement de méta-modèles spécifiques au domaine du contrôle d'accès, capables de représenter l'information de contrôle d'accès de chaque composant de manière abstraite, concise et plus facile à analyser et à comprendre.

Une fois que le métamodèle et les modèles PIM sont disponibles, les opérations d'analyse et de manipulation peuvent être réutilisées. Une application immédiate serait l'utilisation du langage de requêtes standard OCL[71] (Object Constraint Language) pour calculer des métriques intéressantes sur le modèle et effectuer des requêtes avancées sur les règles de contrôle d'accès qui y sont représentées. A titre d'exemple nous pouvons facilement compter le nombre d'autorisations par sujet, ainsi qu'interroger si un sujet donné dispose d'autorisations sur un objet donné, etc. Chacune des opérations définies pourraient être appliquées à tout modèle PIM, peu importe le système d'où l'information a été extraite. D'autres tâches d'analyse, comme la détection des erreurs de configuration et des anomalies dans les politiques, peuvent également être réutilisées à ce niveau.

1.3.3 Modèle des exigences CIM

Les modèles PIM représentent des informations d'une manière abstraite, de sorte que les détails concernant les spécificités de la plate-forme de mise en œuvre sont éliminés. Néanmoins, les modèles PIM représentent toujours l'information par rapport à un domaine donné, c'est à dire qu'ils comprennent les concepts et les relations qui ne sont pertinents qu'à ce domaine. L'utilisation de ces constructions spécifiques à un domaine facilite la compréhension de la politique de sécurité lorsqu'elle est analysée dans le contexte du domaine (à titre d'exemple, pour un expert en sécurité de bases de données, un PIM doit comprendre les concepts de la table, colonne, etc. et les relations entre eux). Cela facilite la compréhension de la spécification de contrôle d'accès.

Cependant, bien adaptées à plusieurs domaines différents, les politiques de contrôle d'accès sont un mécanisme qui ne dépendent pas d'un domaine concret. Ainsi, comme un pas additionnel, les modèles spécifiques à un domaine, peuvent être extraits vers un modèle de contrôle d'accès générique CIM.

Comme avec la transformation de PSM à PIM, qui a permis de traiter des modèles de contrôle d'accès spécifiques à un domaine d'une manière uniforme sans tenir compte des spécificités de la mise en œuvre et des fournisseurs, la traduction de PIM à CIM permet de manipuler des modèles de contrôle d'accès extraits de différents domaines d'une manière uniforme et réutilisable. Il permet également l'intégration de tous les modèles PIM en un seul CIM.

Nous voulons remarquer que, bien que la traduction de PIM à CIM présente plusieurs avantages pour la manipulation des politiques, elle peut avoir pour conséquence que les politiques de domaines concrets soient plus difficiles à comprendre, étant donné que les concepts PIM peuvent être perdus dans la traduction. Idéalement, le langage cible CIM doit fournir les mécanismes (parmi les mécanismes possibles nous avons les stéréotypes, les valeurs marquées, l'extension du modèle, etc) d'intégrer ces concepts spécifiques au domaine, de façon qu'une politique représentée dans la langue CIM soit à la fois, indépendante de calcul et indépendante du domaine pour les tâches de manipulation et d'analyse, mais aussi spécifique au domaine, de sorte qu'il soit plus facile à comprendre par les experts du domaine.

Grace à sa flexibilité (il est capable de représenter les politiques RBAC, mais aussi des politiques avec d'autres modèles) et à son extensibilité (il peut donc intégrer les concepts spécifiques au domaine). Nous croyons que XACML[59], langage et un framework standard pour la représentation et la gestion des politiques de contrôle d'accès, est un bon candidat pour la représentation des politiques issus de différents composants.

Intégration des politiques de contrôle d'accès

Comme indiqué précédemment, les politiques de contrôle d'accès des composants utilisés pour construire un système complexe doivent collaborer pour atteindre des exigences de sécurité. Ainsi, des interactions entre les politiques existent. Ainsi, la dernière étape de notre framework consiste à combiner les politiques provenant des différents composants du système, de sorte qu'une information utile puisse être obtenue et les propriétés, par rapport à une politique globale, puissent être vérifiées. Nous devons être en mesure de vérifier si une opération qui n'est pas autorisée dans un composant est autorisée dans un autre ne crée pas une vulnérabilité dans le système. Nous devrions être aussi en mesure d'assurer que lorsqu'un sujet obtient une autorisation sur un objet donné, tous les composants participent correctement de sorte que l'objet peut être atteint par le sujet comme souhaité.



Introduction

Most company's information systems are composed by heterogeneous components, responsible of hosting, creating or manipulating critical information for the day to day operation of the company. Securing this information is thus a main concern, imposing the system to meet diverse security requirements. In this sense, availability, confidentiality or integrity properties, among many others, may be demanded requirements for the proper operation over the data managed by any given information system.

With the purpose of assuring these security properties, both, researchers and tool vendors have proposed and developed during the last decades a plethora of security mechanisms aimed at ensuring that the data of an information system (and the systems itself) is safe from possible threats. Moreover, many of those mechanisms have been integrated in concrete information system components (as databases, operating systems, web servers, etc) making security a core component of many of them.

Among all the security properties an information system must require, confidentiality and integrity, the security properties assuring that 1) information is available only to authorized parties and 2) Information is accurate and consistent, are two of the most critical ones. Failing to assure any on those properties when required, will lead to a compromise in the valuable information asset that may cause important losses to the company. Due to its relatively conceptual simplicity, one of the most used mechanisms in information system components to assure data confidentiality and integrity are *access control (AC)* policies. Grosso modo, Access-control policies establish for a given system a set of rules indicating which actions a given actor in the system can perform on an information resource.

However, and despite the few methods that attempt to derive these policy implementations from high-level security specifications, the task of implementing an access control security policy in concrete components remains in the vast majority of cases complex and error prone as it requires knowing low level and vendor specific tools and techniques. Moreover, in the case of complex systems composed of a number of interacting heterogeneous subsystems, access-control is pervasive with respect to their architecture. We can find access-control enforcement in different components placed at different architectural levels. Therefore, in any system, a set of different access control policies are enforcing the security goals. But these policies are not independent and relations exist between them, as relations exist between components situated in different architecture layers.

Thus, ideally, a global representation of the access-control policy of the whole system should be available, as analysing a component policy in isolation does not provide enough information. However, normally, this global policy only exist in an implicit and not always consistent manner.

In this context, discovering and understanding which security policies are actually being enforced by the information system comes out as a critical necessity. This is a necessary condition for the reengineering of the current policies to adapt them to evolving needs of the company and also to detect inconsistencies between the enforced policies and the desired policies. The main challenge to solve is bridging the gap between the vendor-dependent security features and a higher-level representation that express these policies in a way that abstracts from the specificities of specific system components and that can be understood by security experts with no deep knowledge of the particularities of each technology and vendor. This logical model would also allow us to implement all evolution/refactoring/manipulation operations on the security policies in a reusable way.

2.1 Problem description

As introduced above, Access-control policies are often the mechanism of choice for the purpose of assuring confidentiality in information systems. However, the implementation of such mechanisms remain complex and error prone and therefore, given a complex information system where several components implement and enforce access-control policies, two critical questions arise:

1. Does the policy deployed in a concrete system meet the security goal, e.g., does the deployed policy enforces confidentiality and integrity as desired?
2. When working in a complex system encompassing them, are the component policies consistent with each other so that 1) The security global goal

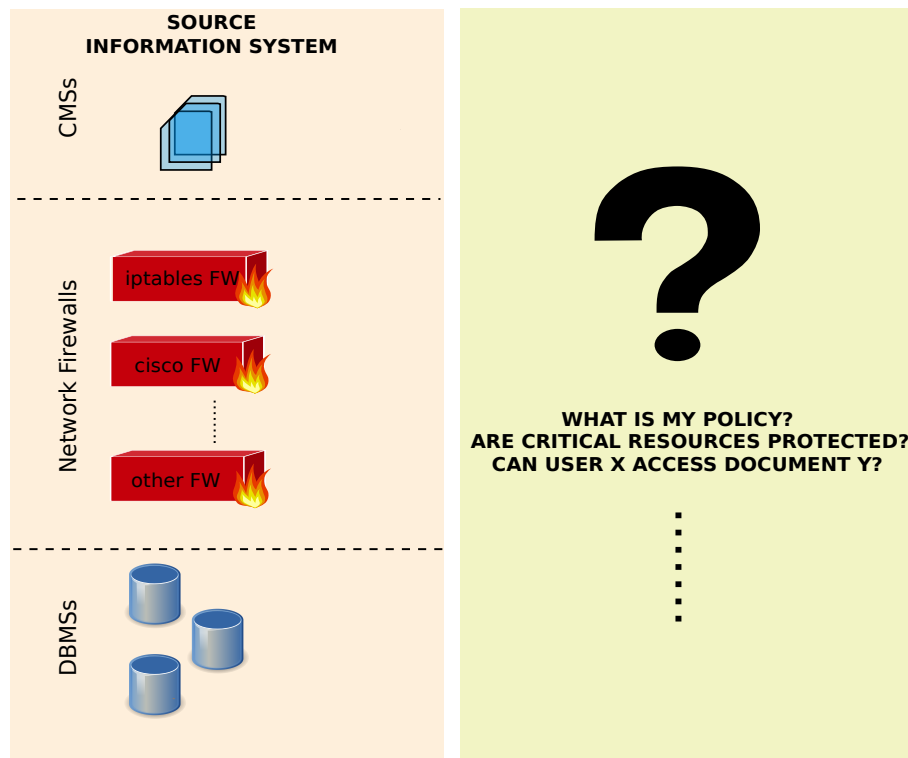


Figure 2.1: The discovery problem

is achieved and 2) The functioning of each component is not unexpectedly impacted by other component's policies?

Unfortunately, answering the aforementioned questions is a very complex task. This is due mainly to two factors. 1) The problem of understanding the policies implemented in concrete systems while already deployed 2) The problem of analysing all the policies in an information system as a whole.

2.1.1 Example

In order to ease the discussion and evidence the aforementioned problems, we will introduce an example of a simple yet quite common information system where several components enforce access-control policies.

Figure 2.2 represents a very common information system configuration. It is composed by a network layer and a number of servers providing diverse services. Among them, we have a database server, that can work standalone but also serves as a data back-end for other services in the application layer. In the example, one prominent element of the application layer is a Content Management System. The network layer, the database storage back-end and the CMS, all implement access-control policies in order to protect the data they manage (notice that this example will be extended in the chapters devoted to the concrete components and system

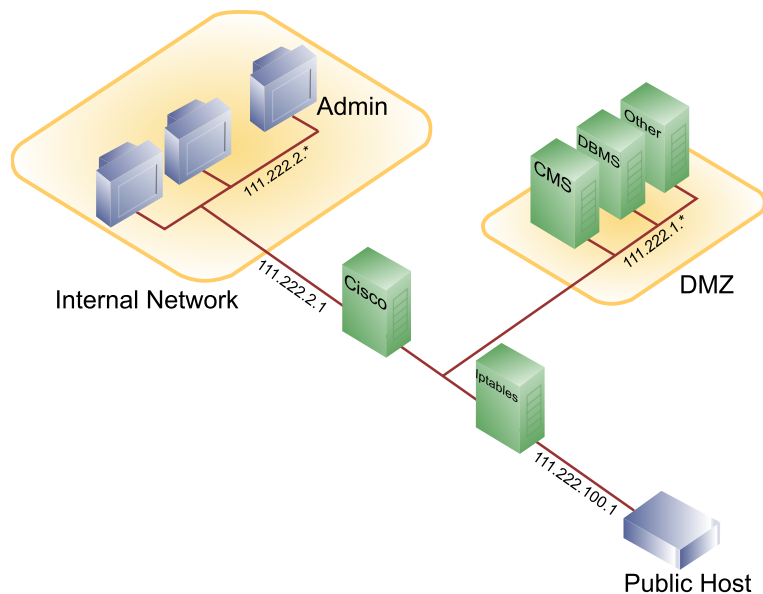


Figure 2.2: Information System Example

analysis).

Thus, we have three different components building up an information system and enforcing access-control policies. In the following, we will show the problems we face when dealing with them individually and with the information system as a whole.

2.1.2 The problem of understanding component policies

Regarding each concrete component, the very complexity of the implementation tools and techniques has a strong impact not only while developing the policy but also when it needs to be inspected for evaluation. Final policy configurations are commonly represented by using low-level representations, e.g., text files with vendor-specific languages, or databases dictionary tables, again following different data schema's from one vendor to another. Moreover, several different mechanisms can participate in the implementation of the policy, scattering the policy and increasing the complexity of the implementation and inspection process.

As a consequence, the errors that may have been introduced during the implementation phase, making the implemented policy hold differences with respect to the desired one, are difficult to detect. Furthermore, as security requirements are rarely static (new application scenarios, new users, etc), frequent modifications of the security policy implementation are required, what increases the chances of introducing new errors and inconsistencies as it can not be easily analysed.

In order to better illustrate this problem, we analyze here the AC implementation of the concrete components of our example. In the following, the AC implementation of networks, RDBMSs and CMSs is briefly explained.

Network

In the network level, access control is enforced by firewalls. The implementation of access-control policies in these components remains a complex and error prone process usually performed by hand relying in the expertise of the system administrator.

Listing 2.1: Firewall 2 Netfilter Iptables configuration

```
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP

iptables -N IntraWeb_HTTP
iptables -A FORWARD -s 111.222.2.0/24 -d 111.222.1.17 -p tcp --dport 80 -j
IntraWeb_HTTP
iptables -A IntraWeb_HTTP -s 111.222.2.1 -j RETURN
iptables -A IntraWeb_HTTP -s 111.222.2.54 -j RETURN
iptables -A IntraWeb_HTTP -j ACCEPT
```

Listing 2.2: Firewall 2 FreeBSD IPFW configuration

```
# Rule Fw2Policy 3 (global)
"$IPFW" add 40 set 1 drop tcp from 111.222.2.1 to 111.222.1.0/24 80
|| exit 1
#
# Rule Fw2Policy 4 (global)
"$IPFW" add 50 set 1 drop tcp from 111.222.2.54 to 111.222.1.0/24 80
|| exit 1
#
# Rule Fw2Policy 5 (global)
"$IPFW" add 60 set 1 permit tcp from 111.222.2.0/24 to 111.222.1.0/24 80
setup keep-state || exit 1
#
# Rule fallback rule
# fallback rule
"$IPFW" add 70 set 1 drop all from any to any
|| exit 1
```

Listing 2.3: Firewall 2 Cisco PIX configuration

```
!
! Rule Fw2Policy 3 (global)
access-list eth1_acl_in remark Fw2Policy 3 (global)
access-list eth1_acl_in deny tcp host 111.222.2.1 111.222.1.0 255.255.255.0
eq 80
!
! Rule Fw2Policy 4 (global)
access-list eth1_acl_in remark Fw2Policy 4 (global)
access-list eth1_acl_in deny tcp host 111.222.2.54 111.222.1.0 255.255.255.0
eq 80
!
! Rule Fw2Policy 5 (global)
access-list eth1_acl_in remark Fw2Policy 5 (global)
access-list eth1_acl_in permit tcp 111.222.2.0 255.255.255.0 111.222.1.0
255.255.255.0 eq 80
```

```
access-group eth1_acl_in in interface eth1
```

Listings 2.1, 2.2, 2.3 are firewall configuration excerpts showing the rules needed for managing the ability of sending HTTP requests from the local network through the first firewall in Figure 2.2. All them implement the same policy, namely, allowing all hosts to send HTTP requests except for an specific admin host (111.222.2.54) and the firewall interface itself (111.222.2.1) but they are written by using three different packet filter languages. 2.1 uses Netfilter Iptables, 2.2 uses FreeBSD Ipfw language and 2.3 corresponds to the Cisco PIX filter language.

In order to understand which policy is being implemented, for each kind of firewall the language used for the implementation of the policy and its execution semantics need to be mastered. Moreover, as a number of (possible different) firewalls can collaborate in enforcing the policy in a given network, the policy may be scattered with respect to a given network topology, increasing the complexity of the understanding process, as rules for the same services and hosts can be present in several different configuration files.

Database Management Systems

Due to the key role Database Management Systems play in information systems for storing required, often critical information, access-control models and techniques have been early adopted by database vendors. Moreover, the SQL standard already provides the basic means to manage permission on databases (although not all vendors strictly follow the standard).

Unfortunately, as in the case of the network layer, understanding which policy is being implemented is challenging due to a number of reasons:

1. First of all, not all vendors follow the same access-control model. While they mostly implement Discretionary Access Control (DAC) policies, i.e., the owner of the resource can delegate the permissions, some of them enhance it by adding role-based capabilities.
2. Although the SQL standard provides the means to define a basic access-control policy (e.g., grant and revoke commands), once implemented, it is normally stored in internal dictionary tables, following a vendor-specific schema.
3. Several other mechanisms participate in the implementation of the policy. Concretely, stored procedures and triggers. The latter may be used to implement fine-grained access-control rules while the former may be used to factorize permissions, following a delegation schema. Both these mechanisms, are vendor-specific. Different languages and rules are used depending on the database in hand.

CMSs

If we take a look to the most used CMSs we can observe that access-control in CMSs vary largely. Some support roles while other don't, some provide the user with an static immutable set of predefined roles while others allow the user to define his own roles. The level at which permissions can be granted can also vary. In some systems, the permissions can be granted of object types, while in others, permissions can be granted in individual objects.

Looking closer to three of the most used CMSs, namely Wordpress, Drupal and Joomla, we see how the complexity and flexibility of the AC implementation grows. Wordpress and Drupal present the simplest AC schema, with predefined roles and permissions at the object type level. Joomla allows through the use of groups the implementation of an RBAC access-control policy. Moreover, for all the mentioned CMSs, there exist a vast number of access-control pluggable modules. These modules provide a wide range of features, from complex RBAC policies to fine-grained access-control, increasing the complexity and the differences between the AC implementations of the CMSs.

With all this diversity, and having in mind that migrating from one CMS to another is a common practice, the need for extracting the access-control policy arises as both, a challenge and a requirement. Once implemented, and given the aforementioned complexity and diversity, understanding and analysing the policy in order to asses the absence of errors and facilitate its manipulation is hampered by demanding a deep component-specific knowledge.

2.1.3 The problem of analysing the access-control policies as a whole

We have seen how access-control policies are deployed in concrete components. However, policies can not be regarded as isolated. When working in an encompassing complex systems, concrete components collaborate in order to meet its functional requirements. Thus, the access-control policy on a given component may be affected by the access-control policy on another component. Together, they are supposed to collaborate to achieve the global security goal. Unfortunately, the relations between the components and their access-control policies are rarely explicit, what prevents them to be analysed as a whole. Supposing the component C_x depends on the component C_y for its functioning, the schema of permissions in C_y , while correct when analysed in that context, may put at risk the component C_x or prevent it to work properly.

In our example, dependencies exists between the information system components. The Database depends on the network while the CMS depends on both the

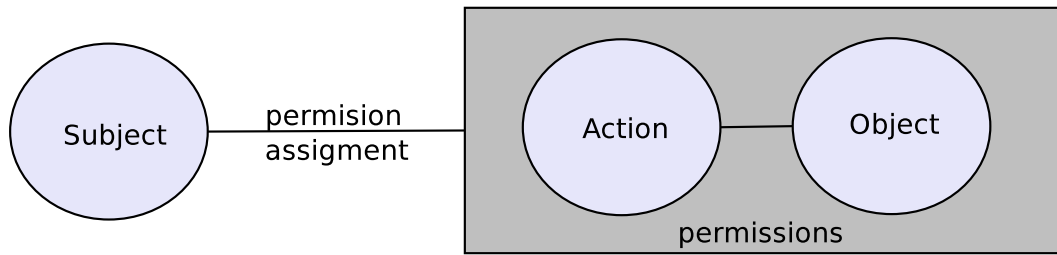


Figure 2.3: Basic Access-Control

network and the Database. In this scenario, the combination of their access-control policies may lead to unexpected behaviour. Access to a given resource may be unexpectedly denied (or allowed). As an example, a combination of permissions in the network and the database policies could lead to the disclosure of CMS data, supposedly protected by its AC policy.

2.2 Basic Concepts

This thesis aims to solve problems in the domain of information security by applying model-driven tools and techniques. In order to ease the discussion, in this section the basic concepts of these domains, relevant to our work, will be summarized. Concretely, we will provide background on access-control policies, model-driven core concepts and model-driven reverse engineering techniques.

2.2.1 Access-Control

Access-control [86, 19], often simply called *Authorization* or *Secrecy*, is a mechanism aimed at assuring that the information within a given Information System (IS) is available only to authorized parties. Therefore access-control it is used in order to assure two system properties, Confidentiality, but also Integrity, by controlling that only trusted entities modify or write the data.

In Figure 2.3 the core concepts of access-control, namely Object (or Resource), Subject, Action and Permission (or Privilege) are depicted. The description of these concepts is as follow:

- **Object:** Objects are normally passive entities within systems. They represent pieces of information such as files in operating system or tables in relational databases. However, sometimes these object can also represent running services like HTTP servers, etc. Thus, objects in the context of access-control are any resource that can be accessed within an Information System.

- **Subject:** Subjects are the active entities in a system. They represent the actors to which the access to *Objects* is controlled. Again, as with objects, these actors could be, not only human users, but also machines or services that access other services, etc.
- **Actions:** Actions are any kind of access to the *Objects* that may be performed by the *Subjects* in a given IS. From the classical C.R.U.D. (Create Read, Update, Delete) operations in database systems to sending a HTTP package in a network.
- **Permission:** *Actions* are related to *Objects*, constituting the permissions. A permission is thus the right to perform a given *Action* (or set of actions) on a given *Object* (or set of objects). These permissions are, in turn, granted or denied to the *Subjects*.

Summarizing, access-control is about granting or denying to *Subjects* in a system the *Permissions* to perform *Actions* on *Objects*. The definition of the permissions and its assignment is performed by using two concepts:

- **Rule:** A rule is the assignment (or denial) of a permission to a given subject. Generally, access control rules have the following form:

$$R_i : \{conditions\} \rightarrow \{decision\},$$

where the subindex i specifies the ordering of the rule, *decision* can be accept or deny and *conditions* is a set of rule matching attributes like the source and destination addresses, holding roles, but also environments conditions like time, etc.

- **Policy:** An access-control security policy is the set of permission assignments within a given information system. Thus, it is composed of a set of Rules. This policy constitutes a mere definition of the security requirements for the system, while the process of implementing the mechanisms to make the system follow the rules it defines is called enforcement.

The core concepts of access-control described above can be organized in different ways and hold different meanings. Concretely, Access-control policies are defined conforming to access-control models that define the elements necessary to construct their contained rules along with their semantics. Due to the efforts of both, the research and industrial communities, have committed to the subject, diverse models have been proposed in the last decades.

In the following, we will briefly describe the most relevant ones.

Access-control Models

AC Lists: Access Control Lists (ACLs) are the most basic form of access control. The concept of an ACL is simple: each resource on a system to which access should be controlled, has its own associated list of mappings between the set of entities requesting access to the resource and the set of actions that each entity can take on the resource. For example, a stateless packet filter firewall follows this model, holding rules that filter the traffic depending on ip addresses, etc. Also, each file on a file system might have an associated data structure that holds the list of users that the operating system as a whole recognizes, along with a flag which indicates whether each user may read, write, execute, delete, or modify the file (or some combination of these).

Mandatory AC: [8]: Mandatory Access Control (MAC) is a type of access control in which access to objects is restricted based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e., clearance) of subjects to access information of such sensitivity. The administrator defines the usage and access policy, which cannot be modified or changed by users. The complexity of this model has reduced its implantation to environments with highly critical information, like military or medical information.

Discretionary AC: Discretionary Access Control (DAC) (also defined in [8]) is a type of access control in which a user has complete control over all the programs it owns and executes, and also determines the permissions other users have those those files and programs. We have examples of DAC implementations in the domain of relational databases and file systems where the owner of a table or a file can grant different permissions on it to other parties.

Role-based AC: Role-based Access Control (RBAC) [85] is a newer access control model than the ACL, MAC and DAC paradigms. The basic idea behind this model is the use of roles. Permissions are established based on the functional roles in a given enterprise and then users are assigned to roles. In other words, the requester's role or function will determine whether access will be granted or denied. Being that roles represent the tasks and responsibilities within an enterprise, they are less variable than users, simplifying the implementation and administration of the policy.

Depending of which level of the RBAC model is under consideration, it may include some more advances concepts. Role-hierarchies, sessions, separation of duties and fine-grained constraints can be found in the highest RBAC model level (see Figure 2.4).

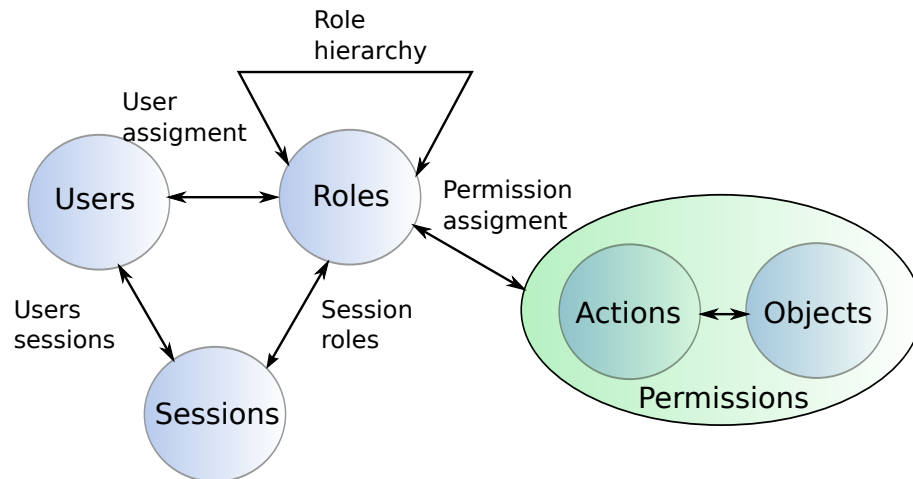


Figure 2.4: Attribute-Based Access-Control

Organization based access control: Organization based access control (OrBac) [9] presents an extension to the RBAC model specially tailored to the definition of security policies independently of the implementation. Subject, Object and Actions, the basic access-control concepts are abstracted to, Roles, Views and Activities. In this manner, the privileges within an organization can be defined by using only the abstract entities, and then, concrete privileges derived by assigning concrete entities to the abstract ones. OrBac is also intended to provide support for dynamic scenarios, thus, it introduces the concept of Context so that assigned permissions depend on the evaluation of context attributes. Finally, OrBac is not limited to permissions but also includes prohibitions, obligations and recommendations.

Attribute-based AC : Attribute Based Access Control (ABAC)[99] is an access control model wherein the access control decisions are made based on a set of characteristics, or attributes, associated with the requester, the environment, and/or the resource itself (see Figure 2.5). Each attribute is a discrete, distinct field that a policy decision point can compare against a set of values to determine whether or not to allow or deny access. The attributes do not necessarily need to be related to each other, and in fact , the attributes that go into making a decision can come from disparate, unrelated sources.

2.2.2 Model Driven Engineering

The Model Driven Engineering (MDE)[23] paradigm emphasizes the use of models to raise the level of abstraction and automation (of the model manipulation operations) in the development of software. Abstraction is a primary technique to cope with complexity, whereas automation is an effective method for boosting productivity (and quality). In MDE, the approach used to increase the level of ab-

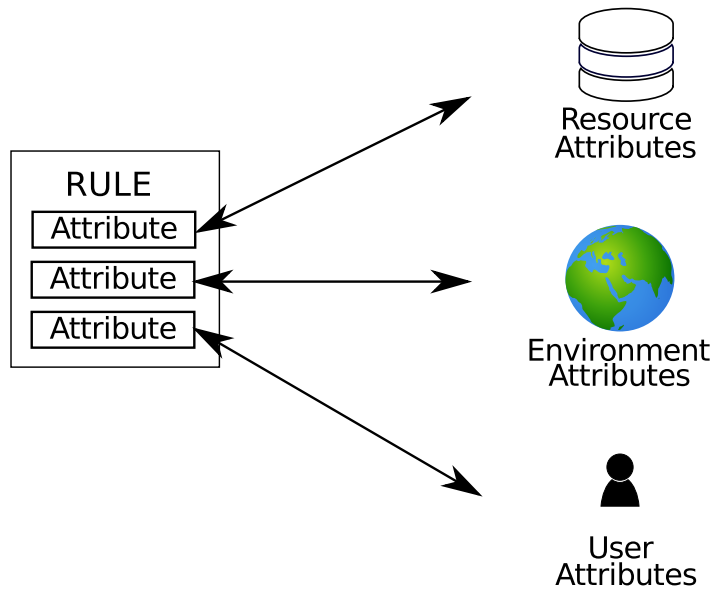


Figure 2.5: Attribute-Based Access-Control

straction is the use of abstract Models, often obtained by defining Domain-Specific Languages (DSLs) whose concepts closely reflect those of the problem domain, facilitating the understanding and hiding the details of the implementation technologies.

The main assumption of MDE is that the model level and not the programming source code is the right representation level for managing all artifacts within a software engineering process. Therefore, models are considered as first-class citizens in MDE. Models are defined according to a three-level architecture shown in Fig.2.6. Such architecture is composed by model, metamodel and metametamodel.

Modeling and MetaModeling

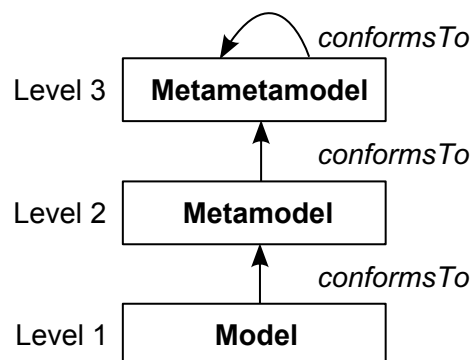


Figure 2.6: Three-level architecture in MDE [35]

A model is a (possibly partial) representation of a system that captures some of

its characteristics. A model contains elements/entities that represent entities composing software artifacts/concepts in the real world. Such concepts and their associations (i.e., semantics) are defined in the second modeling level, called metamodel.

A model is related to a metamodel according to a relation of conformance. Such relation is equivalent to the relation the code written in a given programming language has with respect to the grammar of that language. As programs written in one language must conform to the grammar rules of that language, models defined according to a metamodel must conform to the rules defined in that metamodel.

Metamodel's elements/entities and relations are in turn defined by means of the third modeling level called metametamodel. Similar to the model/metamodel relationship, a relations of conformance is defined between metamodels and metametamodels; such that a metamodel is defined using concepts and associations of a given metametamodel. Again, this relation is equivalent to the relation between the grammar of a given programming language and a language to define grammars (e.g., EBNF: Extended Backus-Naur Form).

Finally, models, metamodels and metametamodels may be implemented according to different modeling standards. For instance, the Object Management Group proposes a standard metametamodel called Meta Object Facility (MOF)[72] and different standard metamodels being, the Unified Modeling Language (UML)[74], the one with the more widespread use.

Model Transformations

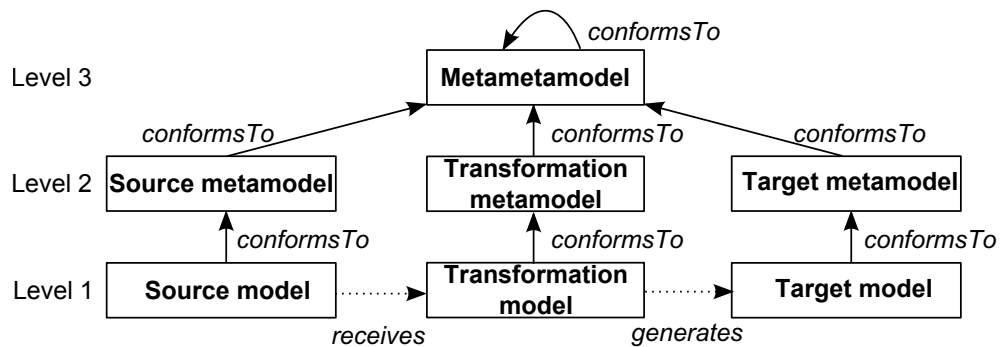


Figure 2.7: Model transformation [35]

The second feature of MDE is the automatic model manipulation. This manipulation is usually performed by means of model-to-model transformations (M2M) (see Fig.2.7) that, taking one or more models as input, generate one or more models as output (note that input and output models not necessarily conform to the same metamodel) according to mappings defined over the concepts of the input and output metamodels.

The MDE paradigm encourages the rigorous use of models and model manipulations in order to deal with the complexity of systems. However, the MDE

world needs to collaborate with other existing technologies. This relation is also performed by means of model transformations. Text-to-Model (T2M) and Model-to-Text (M2T) transformations aim at bridging the gap between MDE and other technologies. We call the different technologies we can find when dealing with information systems, Technical Spaces (TS). The concept of technical space is key to the works of this thesis and therefore it will be described in detail (based on the description that can be found on [29]) in the following section.

Technical Spaces

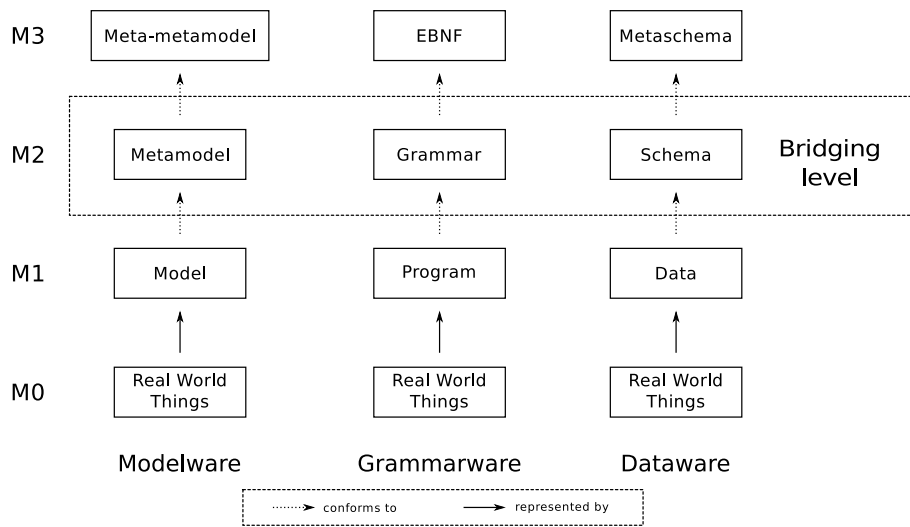


Figure 2.8: Bridging technological spaces [29]

The concept of Technical Space is introduced in [57]. A TS is defined as a working context with a set of associated concepts, body of knowledge, tools, required skills, and capabilities. The artifacts of which a software system is composed conform to the TS used to develop them. For example, as introduced before, the source code of a program conforms to the grammar TS, called grammarware, whereas XML documents conform to the XML TS, that we can call xmlware. A TS is defined based on a number of artifacts in different abstraction levels which are related to each other by means of a conformance relationship (e.g., program/grammar in grammarware, xml document/schema in xmlware and model/metamodel in modelware).

However, TSs are not isolated and bridges can in fact be defined between two different TSs. Bridging TSs allows the artefacts created in one TS to be transferred to another different TS in order to use the best capabilities of each technology, promoting interoperability between applications.

When bridging the MDE TS, called modelware, with other TSs, two operations should be supported: (1) the extraction of models from software system artefacts

defined in the TS considered and (2) the generation of these artefacts from models. For instance, a bidirectional grammarware-modelware bridge should provide both a model extractor from source code and a source code generator from models. *t2m* and *m2t* transformations applied to source code files and models enable the definition of one-way bridges between grammarware and modelware.

In M2M transformations, both the source and target elements of the transformation are inside the modelware technical space. However both M2T and T2M must deal with the different artifacts of which a software system is composed, that is, they must deal with the TS to which the artifact involved in the transformation are implemented. In the case of M2T transformations, the vast majority of existing solutions allow these transformations to be defined through the use of templates, where the text to be generated conforms to a layout which is filled in by model information. On the other hand, solutions aimed at performing T2M transformations are normally ad-hoc tools which deal with a specific type of artifacts. The main difference between M2T and T2M is that template based M2T transformations can generate any text-based artifact (i.e., source code, XML les, etc) without regarding the formalism to which such artifact conforms, whereas T2M transformations must know such formalism to be able to produce correct artifacts (correctness from the point of view of the conformance relationship).

There exist specific languages for defining M2M and M2T transformations, for instance, ATL [51], the OMG standard QVT[73], Epsilon¹ for M2M and MOF-Script², JET³ and XPand⁴ for M2T, whereas T2M transformations are normally performed by handcrafted specific solutions. However, some generative solutions for specific TSs like grammarware exist, as it is the case of the XTEXT⁵ framework, a tool that generates the required parsers to get models from language instances by providing the corresponding grammar.

2.2.3 Model Driven Reverse-Engineering

In MDE processes aimed at the creation of new systems, abstract models are initially built by developers to describe the system, and model transformations are then in charge of generating artifacts of the new software system. In model-driven software reengineering[34], a reverse engineering process is first applied to obtain the initial models, which will later be transformed in order to restructure and generate the system evolved. A model-driven reverse engineering process is generally composed of two steps:

-
1. <https://www.eclipse.org/epsilon/>
 2. <http://www.eclipse.org/gmt/mofscript/>
 3. <http://www.eclipse.org/modeling/m2t/?project=jet>
 4. <http://www.eclipse.org/modeling/m2t/?project=xpand>
 5. <http://www.eclipse.org/Xtext/>

1. Extracting low-level models from the existing system artifacts and
2. Transforming the extracted models into high-level ones, which will be used in the rest of the reengineering process.

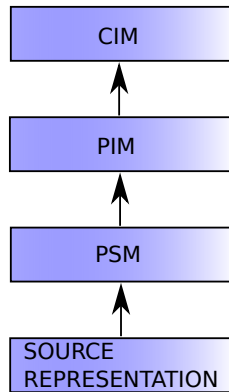


Figure 2.9: Model-driven reverse engineering

In Figure 2.9 the classical model-driven reverse engineering steps are depicted. Starting from the source representation, Platform specific models (PSMs) are extracted, so that we can move from the original technical space to the modelware technical space where the modeling tools and techniques become available. These models are however typically in the same abstraction level of the source representation. The next step is thus to raise the abstraction level of the PSMs, so that the concrete implementation details, vendor-specific features, etc., are eliminated. The results of this process is a Platform-independent model (PIM), able to represent the information of the domain, disregarding low-level, implementation information. Normally, operations will be defined at this level, as the reusability is assured by the platform independence. The abstraction level can be then further abstracted, so that we get rid not only of the platform concrete information but also of the computation domain. In this level, we talk about Computation-Independent Models (CIMs) Note however that it is not mandatory to strictly follow this schema. It is possible to pass directly from source code to PIM, or to avoid the CIM level, etc.

The model extraction step (injection) is therefore a critical step in modeldriven software reengineering as it is the step bridging the gap between the existing system and the modelware.

2.3 State of the art

Our thesis aims to contribute to the domain of security of information systems, by using model-driven techniques. Concretely, we have focused on a high-level requirement, data confidentiality. Access control models are the more common solution for that requirements and as such, we have explored their state of the art

together with the model-driven approaches related with them. As we are following a bottom-up or backward approach, we have also studied the general state of the art of reverse engineering focusing in its interactions with model-driven techniques.

As for the chosen set of initial information system components to work with, i.e., relational databases, firewalls and content management systems, we have also investigated the state of the art regarding their implementation of access-control mechanisms, analysis techniques and reverse engineering approaches.

Finally, they have explored the existing approaches to the problem of integrating different access-control policies in complex systems.

As a summary, the general classification of the state of the art relevant to this thesis is as follows:

- Access control, modeling and model-driven engineering
- (Model-driven) Reverse engineering
- Database Access-Control
- Firewall Security
- Content Management Systems Access-control
- Policy Integration

In the following, we will detail relevant works in the aforementioned areas.

2.3.1 Access control and model-driven engineering

Work on access-control models has been profuse. In [8] two of the most popular access control models: Mandatory Access Control (MAC) and Discretionary Access Control (DAC) are described. Then, [85] describes the Role-Based Access Control (RBAC) model and tries to unify the several models used to implement it by identifying the common features. Important non-standard features are also described. Then they organise the RBAC proposed model in four cumulative compliance levels: flat RBAC, hierarchical RBAC, constrained RBAC and Symmetric RBAC. Based in RBAC, in [9] the authors present an extension to the RBAC model specially tailored to model security policies that are not restricted to static permissions but also include contextual rules related to permissions, prohibitions, obligations and recommendations. This new model is presented by using a formal language based on first-order logic. More recently, in [99] the authors discuss ABAC, an attribute-based access control model highlighting the flexibility it provides and its possibilities to overcome problems present in the other access-control models. In [50], the authors construct an ABAC model that has the features to be configured to do DAC, MAC and RBAC.

The interaction between security and access-control models with model driven engineering has lead to prominent works. Among them, in Secure UML is pre-

sented. [61] It is a modeling language for the model-driven development of secure, distributed systems based on the Unified Modeling Language (UML). The approach is based on role-based access control with additional support for specifying authorization constraints. Similar, but with a broader focus, in [52] the authors present UMLSEC, an extension of UML that allows to express security relevant information within the diagrams in a system specification. UMLsec is defined in form of a UML profile using the standard UML extension mechanisms. In particular, the associated constraints give criteria to evaluate the security aspects of a system design, by referring to a formal semantics of a simplified fragment of UML. [17] presents an approach to automatically generate system architectures from the models, including complete, configured access control infrastructures [56] proposes a new approach to developing and analyzing RBAC policies using UML for modeling RBAC core concepts and OCL to realize authorization constraints. Dynamic (i. e., time-dependent) constraints, their visual representation in UML and their analysis are of special interest.

Regarding formal models for representing security in systems, the efforts of the research community have been abundant. We discuss here the most relevant ones with respect to our thesis. [18] formally defines how by means of translation from OrBAC to B event Method and by refinement of B models, a system satisfying the given policy can be modeled. Global security needs are mentioned. They work is aimed at working with Confidentiality, Integrity, Availability and Auditability. In [82] the authors follow the path opened in [18]. Focusing in network systems, they intend to validate the deployment of security policies by checking certain security properties. For instance, they define the following formal properties: Completeness (a path exist and the elements of the path have the required functionality for the deployment), accessibility, inaccessibility, integrity and confidentiality. These properties are formally defined for networks and functions to evaluate paths and tunnels exist.

2.3.2 (Model-driven) Reverse Engineering

Reverse-engineering has been proved an useful technique to the recovery of knowledge and modernization of systems. A such, it have been largely studied by the community. Thus, here we list only some representative works, mostly regarding model-driven techniques for performing reverse engineering.

A taxonomy of reverse engineering used terms is presented in [34]. In [87], authors describe a risk-managed approach to legacy system modernization that applies a knowledge of software technologies and an understanding of engineering processes within a business context whereas in [28] an overview of the field of re-

verse engineering is presented, reviews main achievements and areas of application, and highlights key open research issues for the future.

More concretely in the field of model-driven reverse engineering, in [83] the authors introduce the use of model driven techniques to overcome the difficulties classical reverse engineering approaches present such as the lack of standards and time effort quantification while in [98] the problems raised by the evolution of model-based software systems themselves are analysed and challenges to be addressed by research in the area identified. Finally, MoDisco[26], is a prominent model-driven framework intended to make easier the design and building of model-based solutions dedicated to legacy systems reverse engineering.

2.3.3 Network (firewall) security

Modeling firewall configurations has been a widely studied problem. Nearer to our domain, in [80] the authors propose a firewall PIM metamodel represented as an XSD schema. In order to obtain it, features of several firewall vendor languages are analyzed. A comprehensive related work regarding firewall domain specific modelling languages is also provided. The same authors extended their preliminary work and contributed in [81] an improved firewall PIM and a PSM for the Netfilter Iptables packet-filter language. Then, an ATL transformation PIM2PSM and a model-to-text transformation to configuration files are provided. Thus, a complete model-driven forward engineering process to generate firewall configurations is provided. They reuse previous inconsistency and redundancy checking techniques. Moreover, as future work they propose raising the abstraction level representing AC Lists in UML.

In [101], the authors describe a platform-independent representation for firewall rule sets that is used as internal representation for the FWBuilder tool, that generates concrete firewall configurations from abstract representations.

Cuppens et Al. [38] describe how to use OrBac for describing network policies and generate firewall rules. They propose an assignation for roles, subjects and objects whereas each firewall is mapped as a sub-organization. Similarly, in [16] the authors describe Firmato, a tool for the specification and management of firewall rule sets.

Other works, also providing modeling of firewall rule sets are more focused in the analysis of their consistency and correctness. In [100] the authors introduce a static analysis toolkit for firewall modeling and analysis. It applies static analysis techniques to check misconfigurations, such as policy violations, inconsistencies, and inefficiencies, in individual firewalls as well as among distributed firewalls. It is implemented by modeling firewall rules using binary decision diagrams (BDDs).

[88] presents a set of techniques and algorithms that provide automatic discovery of firewall policy anomalies to reveal rule conflicts and potential problems in legacy firewalls, and anomaly-free policy editing for rule insertion, removal and modification. Finally, in [24] the authors model a packet-filter by using the Higher Order Logic framework, Isabel.

Not providing an explicit model for the representation of firewall configurations but working with synthetic rules, in [40] the authors describe a complete set of anomalies for firewall rule sets (namely, shadowing, redundancy and irrelevance) along with algorithms to detect and correct them.

With respect to the recovery of knowledge from already deployed configurations, in [69] a method and tool to discover and test a network security policy is proposed. The configuration files along with the description of the network topology are used to build an internal representation of the policy that can be verified by the user through queries in ad-hoc languages. [64] proposes a method and tool (FANG) to discover and test the global firewall policy. FANG collects and reads all the relevant configuration files, and builds an internal representation of the implied policy and network topology. A model of the net topology should be provided while the configuration files are automatically analysed by parsing. Fang is intended to simulate the recovered network configuration through the use of queries. In [21] (Technical Report) The authors describe a theoretical approach of the deployment, analysis and recovery of firewall configurations. It applies a bi-directional method of enforcing and reverse-engineering system and infrastructure policy. Uses a platform-independent intermediate policy representation in the form of a byte-code like language (like AST), not focused in any implementation. The language they propose should be then assembled into machine-dependant specific commands following this schema: High Level Language IPR configuration commands. Tongaonkar et al., [94] propose a technique that aims to infer the high-level security policy from low level representation, concretely from the rules on firewalls (just packet filtering rules). A merging algorithm is used to extract classes (types) of services hosts and protocols. They end up with more abstract rules.

Finally, regarding the modeling and analysis of stateful firewalls, some approaches aim at describing stateful firewall models [46], while others provide straightforward adaptations of management processes previously designed for stateless firewalls [27]. In [37], the authors uncovered a new type of misconfiguration, denoted as intra-state rule misconfiguration.

2.3.4 Database Security

Access-control mechanisms have been early adopted in commercial relational database engines. In [20] presents a review of security concepts on research and commercial databases. It is mostly focused on access control mechanism: RBAC, MAC, DAC and its incorporation into commercial products.

Regarding the deployment of access-control policies for databases from high-level specifications, [91] represents security aspects at the logical level for datawarehouses by using the Common Warehouse Metamodel (CWM) metamodel. [70] proposes a method to derive a concrete database-based access control implementation out of RBAC policies defined by using a graphical interface and [15] proposes a method to transform temporal RBAC policies, specified in a logic-based notation, into PL/SQL code.

As for the reverse-engineering approaches on relational databases, a good amount of works have been contributed. We describe the most relevant ones.

In [62] the authors propose both a general framework and specific techniques for file and database reverse engineering, i.e. recovering its conceptual schema. The framework relies on a process/product model that matches formal as well as empirical design procedures. Chaing et al. [33] present methodology for extracting an extended Entity-Relationship (EER) model from a relational database through a combination of data schema and data instance analysis. Also extraction EER schemas, in [79] the authors describe a technique that supports (EER) schema extraction from an operating relational database. The method starts from the database schema as stored in the DBMS dictionary, i.e., the relation names, the attribute names and their basic characteristics (uniqueness of value, not null values). Finally, semantics extraction is supported by available queries analysis.

Also, bringing the reverse engineered schema to other domains, [14] proposes a novel approach to reverse engineering of relational databases to ontologies. The approach incorporates two main sources of semantics: HTML pages and a relational schema.

Regarding the reverse-engineering of security aspects, in [89] the authors present an approach to discover and resolve anomalies in MySQL access-control policies by extracting the access-control rules and representing them in the form of Binary Decision Diagrams. The access-control policies are extracted from the database dictionary tables, not including other implementation mechanisms.

2.3.5 Content Management Systems

We can consider the popularity and widespread of Content Management Systems as a quite recent phenomenon. As a consequence, there are not many works in

the literature analysing its security regarding access-control techniques, the modeling of the domain or its reverse engineering.

Regarding the modeling approaches, although not focused in CMSs nor in the access-control aspects, in [32] the authors present a language to define web applications.

Some tools for checking the configuration of WCMSs have been provided and analysed by the scientific communities. However, these tools are focused in low-level security aspects like management of cookies or prevention of SQL injection vulnerabilities [66, 97]. Nearer to the extraction of high-level model representations of access-control, an approach for extracting AC information from dynamic web applications source code is presented in [44, 10].

2.3.6 Policy Integration

The problem of integration of access-control policies belonging to different components, different architecture layers or different authorization authorities have been recognized as a challenge by the security research community.

From the formal point of view, In [31] the authors provide the foundations of a formal framework able to represent policies in different architectural layers and the dependencies between layers. Similarly, in [22] the authors analyze diverse combination needs for access control policies including the combination of hierarchical policies through refinement. Algebraic operations for representing and manipulate these combinations are provided. In [82] Method-B is used to formalize the deployment of access control policies on systems composed by several network components. The authors also define and implement security properties to check the correctness of the deployment process.

Using model-driven techniques, in [39] the authors formalize what they call the policy continuum model. This model is able to represent policies at different inter-related abstraction layers.

In [65] the authors describe some interesting results regarding the integration of policies not belonging to the same authorization entity by relying in the calculation of policy similarity.

Not focused on access-control, [30] provide an approach to detect conflicts between different kinds of policies in the same environment A.C policies, copyright policies, etc.

2.4 Thesis Structure

This thesis is structured as follows. The next chapter presents the objectives and contributions of our thesis whereas an overview of our proposed our method is pre-

sented in Chapter 4. Chapter 5 shows the application of this method to the domain of packet filtering firewalls in networks layers while chapters 6 and 7 do the same for the domains of relational databases and content management systems respectively. Chapter 8 presents our approach for the integration of the models extracted from concrete components into a global model along with analysis facilities and techniques for the detection of multilayer misconfigurations. Chapter 9 discusses related work. Finally, Chapter 10 presents some conclusions and future work.

Some of the results of this thesis have been already published in [63], [77], [76], [78] and [43].

Objectives and Contributions

As shown in Chapter 2, despite the existence of approaches based on formal refinement techniques, e.g., using abstract machines grounded on the use of set theory and first order logic, that ensure, by construction, cohesion, completeness and optimal deployment, the complexity and incompleteness of those methods makes policies to be often empirically deployed over the concrete components based on security administrator expertise and flair. This task, requiring often low level and vendor-specific knowledge, is complex and error prone, evidencing the relevance of the analysis of the already deployed policies, in order to update, detect and correct errors, improve quality features, refactor, etc.

Unfortunately, the extraction and reverse engineering of already deployed access-control policies is also an unsolved challenge. Although a few methods and efforts exists (for some concrete components), they fall short either when dealing with real deployed policies (just focusing in the analysis part and disregarding the extraction step) or by not taking into account complex systems, composed by a number of interacting subsystems.

In this sense, the objectives of this thesis are as follows:

- Choose a number of relevant information system components, lying in different architecture layers and implementing/enforcing access-control policies.
- Propose a general methodology, applicable and adaptable to any information system, allowing for the extraction of a component-specific representation

of the access-control security policy they implement/enforce in a way that it is easier (for the domain experts) to analyse, understand and manipulate than the actual implementation.

- For each component: 1) Define the corresponding component-specific abstract and platform independent representation of the policy. 2) with this extracted access-control policy representation, show how useful tasks can be performed on it.
- Components collaborating in the achievement of the functionality goals of a complex information system also collaborate in the enforcing of the security access-control goal when they implement access-control policies. Thus, one of the objectives of the thesis is to, starting from the extracted component-specific policies of each component, provide the means to build a global representation of the security policy all subsystems help to enforce making explicit the relations between them.
- The last objective of the present work is to, with the global policy representation available, provide the basis (operations, infrastructure, algorithms, etc) for a global analysis of the access-control enforcement of a given system.

In light of the objectives listed above, the first step has been to choose an appropriate set of systems implementing and enforcing access-control policies, so that the framework proposed in this thesis could be demonstrated. For that purpose, we have chosen three different systems. Networks (concretely, firewalls controlling networks), database management systems and content management systems. We want to notice however, that the results of this thesis can be extended to include other different subsystems.

These three different chosen systems work in different architecture layers and often are composed to build up more complex systems. As shown in the example in Chapter 2, a system including firewalls to control and manage the network traffic, a database as an storage facility for applications and a content management systems providing direct service to the user is a quite common configuration. Moreover, access-control implementation/enforcing is a core feature in all of them. Firewalls provide access-control in networks by filtering the traffic according to given conditions. RDBMS and CMSs enforce access-control over the data they store (and also over its administration) by storing a number of access-control rules asserting what users can do.

With the set of systems to work with chosen, the contributions this thesis provide are listed below:

For the domain of Access-Control in Networks (see Chapter 5):

1. We have designed a concise platform-independent metamodel able to represent the access-control information contained in firewall configuration files.
2. We have proposed an automatic model-driven extraction approach that produces models conforming to the network access-control metamodel described above. Moreover, we have shown how the design of the metamodel facilitates the structural verification (detection of misconfigurations) of the information contained in the configuration files.
3. For networks including several firewalls, a (reversible) combination process aimed to integrate the individual policies in a single model is provided. This allows the analysis of the network Access-Control policy as a whole.

For the domain of Access-Control in Relational databases (see Chapter 6):

1. We have designed the first platform independent metamodel specially tailored for the representation of access-control policies on databases (see chapter 6). This metamodel includes not only the static information of privilege grants stored in the database dictionary, but also the fine grained constraints defined by procedural code (triggers) and the transitive grants obtained through executable code.
2. We have provided an automatic approach for obtaining models corresponding to the database access-control metamodel described above. The approach complements standard database reverse-engineering approaches by gathering access-control information from the diverse mechanisms used to implement and enforce it in relational databases. The source code of executable units withing relational databases (triggers and stored procedures) is also analysed to extract fine grained constraints while heuristics are provided to prevent irrelevant data (e.g., non security triggers) to polute the extracted information.

For the domain of Access-Control in CMSs (see Chapter 7):

1. We have designed the first platform independent metamodel for Access-control in the domain of CMSs.
2. We have provided a model-driven extraction approach (specially tailored to the Drupal CMS) that automatically produces models conforming to the defined access-control metamodel.

For the combination of diverse access-control policies in multi-layer architectures (see Chapter 8):

1. We have defined a framework to integrate policies from different concrete components collaborating in an information system in a single policy. This framework comprises: 1) The use of a common access-control policy language for representing the policies of each component 2) The extension of the target language to express domain-specific information and 3) the recovery/representation of the implicit dependency relations between them.
2. We provide the means to, starting from the integrated model defined above, perform useful analysis tasks. Concretely, we show how misconfigurations stemming from the interaction of the access-control policies of the subsystems composing a complex information system can be detected.



4

Model-driven Framework for the extraction of AC models

In order to tackle the aforementioned problem, and because of its proved effectiveness for the specification and realization of complex systems, we propose the construction of a model-driven automatic reverse engineering mechanism capable of analyzing deployed security aspects of components (e.g., concrete firewall configurations) to derive abstract models (Platform-independent Models (PIMs)) representing the policy (e.g., network security policy) that is actually enforced over system components. Once these abstract models are obtained, they can be reconciled with the expected security directives, to check its compliance, can be queried to test consistency or used in a process of forward engineering to generate correct security configurations. Finally, they can be joined to represent the global access-control policy enforced in the information system by using a Computation Independent Model (CIM) able to gather all the information represented in the PIM models.

In Figure 4.1 we depict the overview of the proposed approach for the extraction and analysis of access-control policies from heterogeneous information systems. In the following we will briefly describe the steps composing it.

4.1 Platform specific models (PSMs)

The conceptual distance between the source information and the abstract models is usually important, making the work of the injector very complex. Therefore,

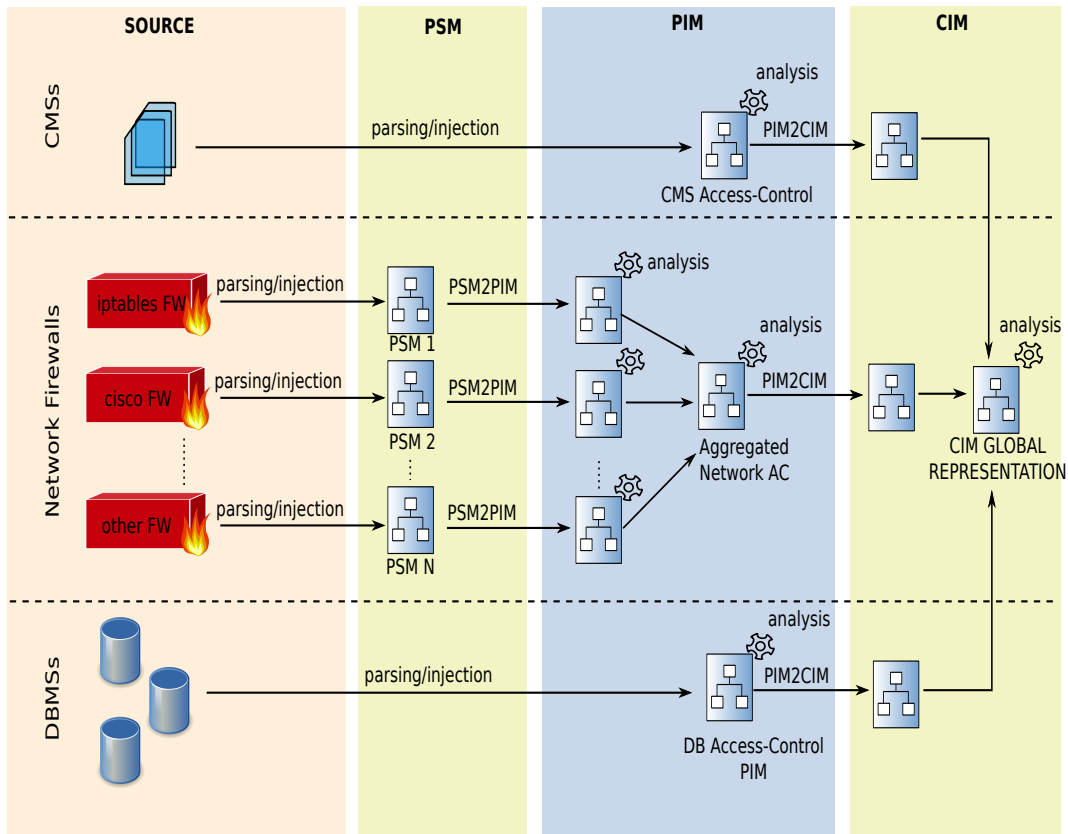


Figure 4.1: Framework overview

the definition of platform-specific models, i.e., models representing the information in the same abstraction level as the source information, may be needed. These models serve as a means to bridge the technical spaces, so that we pass from the technical domain of the source information to the modelware technical space. Once in modelware, the platform-specific models can be transformed into the abstract models by using model transformations tools like ATL[51].

As an example, in the case of firewalls, the AC information is stored in the form of textual configuration files written using, normally, vendor-specific languages with different syntax and semantics. In order to extract the AC information and represent it in an abstract model, parsers and injectors are needed. The parser is able to read the source of the information whereas the injector uses the obtained information in order to fill a model with it. Being the variability and semantic distance of firewall languages w.r.t. to the abstract model big, parsers and injectors will become too complicated. Thus, in order to simplify the task, the AC information is first extracted towards a platform-specific model, specially tailored for the representation of the AC information of a particular vendor. On the contrary, if we take a look to the AC information in relational databases we have a mixed situation. For the extraction of the standard privilege grant information stored in the dictionary tables, there is no need for a PSM, as the concepts are similar to those of the

abstract model. However, for extraction the AC information contributed by triggers and stored procedures a PSM representing the language used for defining the stored procedures is needed, as the distance between the abstract model and the source code is big.

4.2 Extracting abstract (PIM) models

Extracting PSM models creates a bridge between the technical space of the information source and modelware. Having the access-control information of a system represented as a model, enables the reutilization of well-known, off-the-shelf MDE tools. Editor generators, model transformation engines, etc. become automatically available.

However, the information, of the PSMs is at the same abstraction level than the original configuration information. Thus, the next step is to raise the level of abstraction of the information contained in those models. This information is then represented in a Platform Independent Model (PIM), so that it can be manipulated disregarding the specificities of the concrete technologies used for the implementation (notice that when we refer here to platform independence, we intend to emphasize that the models are independent from the concrete vendor-specificities, while still being component specific). Central to this step is the development of AC domain specific metamodels, able to represent the AC information of each component in a abstract, concise, easy to analyse and understand manner.

Once a PIM metamodel and models conforming to it are available, analysis and manipulation operations can be performed in a reusable way. An immediate application would be the use of the well-known OCL[71] query language to calculate interesting metrics on the model and perform some advanced queries on the AC rules represented in it. As an example, we can easily count the number of permissions per subject as well as querying if a given subject has permissions on a given object, etc. Each of the defined operations could be applied to any PIM, no matter what was the concrete system the information was extracted from. Other analysis tasks, like the detection of misconfigurations and anomalies in the policies can also be defined in this level in a reusable way.

4.3 CIM

PIMs represent information in an abstract way, so that the details regarding the specificities of the implementation platform are eliminated. Nevertheless, PIM

models still represent the information w.r.t. a given domain, i.e., they include concepts and relations that are only relevant to that domain. The utilisation of those domain-specific constructs eases the understanding of the security policy when analysed in the context of the domain. As an example, for a database security expert, a PIM including the concepts of table, column, trigger, etc and the relations between them, eases the comprehension of the access-control specification. Same with the domain of content management systems and networks.

However, although adapted and tailored to several different domains, access-control policies is a mechanisms that does not depend of any concrete domain. Thus, as a further step, PIM, domain-specific models can be abstracted towards a CIM, generic, access-control model. We want to notice that, as with the PIM concept, the CIM concept is here adapted to our domain and while it stills represent a high-lever of abstraction w.r.t. the PIM, CIM models in the present work are meant to abstract from the components details (and not from the computation details as in the general case).

As with the translation form PSM to PIM, that allowed to treat domain-specific access-control models in an uniform way disregarding the specificities of concrete implementation and vendors, the translation from PIM to CIM allows to manipulate access-control models extracted from different domains in an uniform and reusable way. It also enables the integration all PIM models in a single CIM.

We want to notice that, although the translation to a CIM brings several advantages to the manipulation of the policies, it may cause policies of concrete domains to be more difficult to understand, as PIM concepts may be lost in the translation. Ideally, the target CIM language should provide the mechanisms (among the possible mechanisms to do this we have stereotypes, tagged values, model extension, etc) to integrate those domain specific concepts, so that a policy represented in the CIM language is both, computational and domain-independent for the manipulation and analysis tasks, but also domain-specific so that it is easier to understand by domain experts.

We believe XACML[59], a standardised, abac-capable, language and framework for the representation and management of access-control policies is a good candidate for the representation of policies coming from different components as it is very flexible (able to represent RBAC policies but also policies with other properties like delegation, etc) and extensible (able thus to integrate domain specific concepts).

4.3.1 AC Integration

As stated in Chapters 2 and 3, the access-control policies of subsystems composing a complex system collaborate to achieve the data confidentiality security goal. Thus, interaction between the policy exist. Thus, the last step of our framework consist in combining the policies coming from the different system components, so that useful information can be obtained and properties with respect to a global policy can be checked. We must be able to check if something not allowed in a component is allowed in another component creating a vulnerability in the system. We should be able to check that when a subject gets a permission on a given object, all components participate properly so that the object can be reached by the subject.

Extracting AC Models from Networks

Firewalls, designed to filter the traffic of a network with respect to a given number of access-control rules, are key elements in the enforcement of network security policies.

Although there exist approaches to derive firewall configurations from high-level network policy specifications[81, 16], these configuration files are still mostly manually written, using low-level and, often, vendor-specific rule filtering languages. Moreover, the network topology, that may include several firewalls (potentially from different vendors), may impose the necessity of splitting the enforcement of the global security policy among several elements. Due to the complexity of the process, it is likely that we end up with differences between the implemented policy and the desired one. Moreover, security policies must be often updated to respond to new security requirements, which requires evolving the access-control rules included in the firewall configuration files.

Therefore, there is a clear need of an easy way to represent and understand the security policy actually enforced by a deployed network system. At the moment, this still requires a manual approach that requires, again, low-level and vendor-specific expertise. Given a network system consisting in several firewalls configured with hundreds of rules, the feasibility of this manual approach could be seriously questioned. While the security research community has provided a large number of works dealing with the reasoning on security policies, succeeding at providing a good analysis and verification of the low-level firewall rules, we believe they fail at obtaining a comprehensive solution as they do not provide a high-level, easy

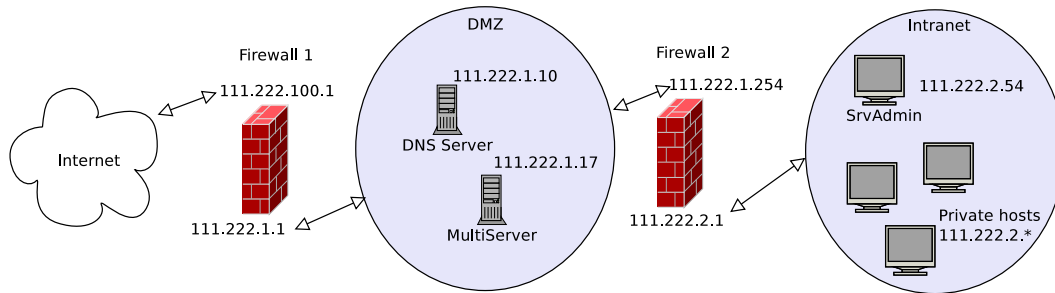


Figure 5.1: Network example

to understand and manage representation nor take, generally, networks composed by several heterogeneous firewalls into account. Moreover, the extraction step is often neglected and the solution presented over synthetic rules without providing the means to bridge the gap between them and the real configurations.

In this sense, we believe that an integrated solution is missing. We believe such a solution must have the following features. First, it has to provide independence from the concrete underlying technology, so that the focus can be put into the security problem and not in implementation mechanisms like chains, routing tables, etc. Second, it has to provide a higher-level representation so that the policy becomes easier to understand, analyse and manipulate. Third, the solution, to be comprehensive, must take into account the contribution of each policy enforcing element (firewall) to the global policy, as the partial picture given by isolated firewalls does not provide enough information to understand the network policy.

In this chapter we propose a model-driven approach aimed at covering this gap. Our approach, first, extracts and abstracts the information of each firewall configuration file to models conforming to a Platform-independent metamodel specially tailored to represent network-access control information in an efficient and concise way. Then, after performing structural verification of the information in the individual models, it combines these models to obtain a centralised view of the security policy of the whole network. Finally, this global network access-control model can be analysed and further processed to derive useful information. As an example, we analyse the structure of its contents to derive the network topology the firewalls operate on.

We validate the feasibility of our approach by providing a prototype implementation working for firewalls using the netfilter iptables and Cisco PIX rule filtering languages. Our prototype can be easily extended to work with any other packet-filtering languages.

5.1 Motivation

In order to motivate our approach, we extend here the Information System example introduced in Chapter 2, subsection 2.1.1, by introducing a second firewall. Let us thus consider we have a De-Militarized Zone (DMZ) network architecture like the one depicted in Figure 5.1.

This is a very common architecture used to provide services both to a local network and to the public untrusted network while preventing the local network to be attacked. It is composed by the following elements:

- An intranet composed by a number of private hosts where one of the private hosts acts as an administrator of certain services provided by the network system.
- A DMZ that contains two servers. A DNS server and a multiserver providing the following services: HTTP/HTTPS (web), FTP, SMTP (email) and SSH.
- Two firewalls controlling the traffic towards and from the DMZ. The first firewall controls the traffic between the public hosts (the Internet) and the services provided by the DMZ. The second firewall controls the traffic between the intranet and the DMZ.

The two firewalls in charge of enforcing the security policy of our example network, could be of the same kind. However, following the *defense in depth*[5] security strategy, it is highly recommended to use two different firewalls so that a possible vulnerability does not affect the whole network. In our example, the *firewall 1* is a linux iptables packet-filtering firewall whereas *firewall 2* is a Cisco firewall implementing Cisco PIX filtering.

In Listing 5.9 we show an excerpt of the configuration file of *firewall 1* with respect to the HTTP and SMTP services. It controls the traffic from the public hosts to the services provided in the DMZ. This sample configuration uses the Netfilter Iptables[84] rule language. Note that this configuration file is written using the Iptables *custom chains* feature, which allows the user to define exclusions to rules without using drop or deny rules.

First, it states in the first three lines that the global policy for the firewall is the rejection of any connection not explicitly allowed. Then, the first chain controls the outgoing SMTP messages towards the public host. It allows them for every host but for the hosts in the local network. The second chain controls the incoming SMTP messages to the server. If the request is done through one machine belonging to the local network, it is rejected while it is allowed for any other machine. The third rule controls the HTTP requests from the public host. Again, connections are allowed for any host but for the local ones.

Listing 5.1: Firewall 1 netfilter configuration

```

iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP

iptables -N Out_SMTP
iptables -A FORWARD -s 111.222.1.17 -d 0.0.0.0/0 -p tcp --dport 25 -j Out_SMTP
iptables -A Out_SMTP -d 111.222.0.0/16 -j RETURN
iptables -A Out_SMTP -j ACCEPT

iptables -N In_SMPT
iptables -A FORWARD -s 0.0.0.0/0 -d 111.222.1.17 -p tcp --dport 25 -j In_SMTP
iptables -A In_SMTP -s 111.222.0.0/16 -j RETURN
iptables -A In_SMTP -j ACCEPT

iptables -N NetWeb_HTTP
iptables -A FORWARD -s 0.0.0.0/0 -d 111.222.1.17 -p tcp --dport 80 -j NetWeb_HTTP
iptables -A NetWeb_HTTP -s 111.222.0.0/16 -j RETURN
iptables -A NetWeb_HTTP -j ACCEPT

```

Firewall number 2 controls the traffic from the private hosts to the services provided in the DMZ. Listing 5.2 shows the rules that control the access to the SMTP and HTTP services. It is written in the Cisco PIX language that does not provide support to a feature like the iptables *custom chains*.

Rules one to six, control the SMTP requests to the server. They are all allowed for the hosts in the private zone discarding only the administrator host, identified by the IP address 111.222.2.54, and for a free-access host, identified by IP address 111.222.2.53. Rules seven to twelve do the same for the HTTP requests. Again, HTTP requests are allowed for all the hosts in the private zone discarding only the administrator host and the free-access host.

Listing 5.2: Firewall 2 Cisco PIX configuration

```

access-list eth1_acl_in remark Fw2Policy 0 (global)
access-list eth1_acl_in deny tcp host 111.222.2.54 111.222.1.17 eq 25

access-list eth1_acl_in remark Fw2Policy 1 (global)
access-list eth1_acl_in deny tcp host 111.222.2.53 111.222.1.17 eq 25

access-list eth1_acl_in remark Fw2Policy 2 (global)
access-list eth1_acl_in permit tcp 111.222.2.0 255.255.255.0 111.222.1.17 eq 25

access-list eth1_acl_in remark Fw2Policy 4 (global)
access-list eth1_acl_in deny tcp host 111.222.2.54 111.222.1.17 eq 80

access-list eth1_acl_in remark Fw2Policy 5 (global)
access-list eth1_acl_in deny tcp host 111.222.2.53 111.222.1.17 eq 80

access-list eth1_acl_in remark Fw2Policy 3 (global)
access-list eth1_acl_in permit tcp 111.222.2.0 255.255.255.0 111.222.1.17 eq 80

access-group eth1_acl_in in interface eth1

```

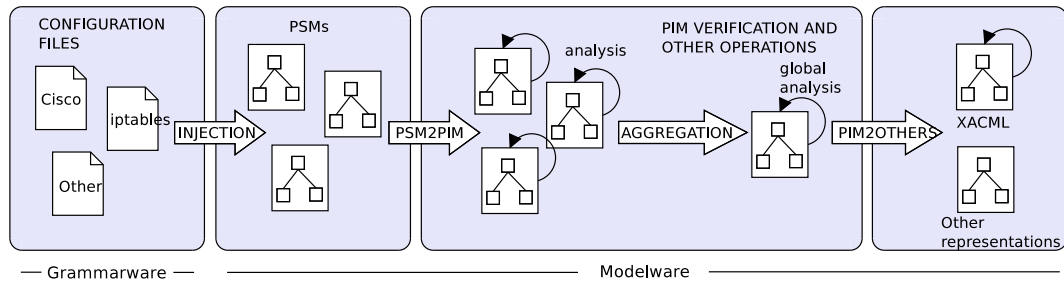


Figure 5.2: Extraction approach

5.1.1 Example Evaluation

Faced with this example, a security expert willing to understand the enforced access control rules will have to directly review the configuration files of the firewalls in the system (disregarding the low-level and often incomplete management tools provided by the firewall vendors, obviously only valid for the firewalls of that vendor), which in this case, involves two different rule languages. Not even the topology picture of the network, provided here with the purpose of easing the discussion, can be taken for granted but instead needs to be derived from the configuration files themselves.

Therefore, we can see that the task of extracting the global access control policy enforced by the set of rules in these two firewalls (that are just minimal excerpts of what a full configuration policy would be) requires expert knowledge about Netfilter Iptables and Cisco PIX. Its syntax along with its execution semantics would have to be mastered to properly interpret the meaning of the configuration files. Moreover, the information from the two configuration files and the default policies would have to be combined as they collaborate to enforce the global policy and can not be regarded in isolation.

In corporate networks potentially composed by up to a thousand firewalls, composed by hundreds of rules and potentially from different vendors using different configuration languages and execution semantics, the task of manually extracting the enforced access control policy would become very complex and expensive, seriously hampering the analysis and evolution tasks the dynamic environment of corporations impose. This is the challenge our approach aims to tackle as described in the next sections.

5.2 Approach

This section details our MDE approach to generate a high-level platform-independent model providing a global view of all access-control rules in a set of firewall configurations files.

Our model-driven reverse engineering approach, that extends the preliminary one in [63], is summarized in Figure 5.2. It starts by injecting the information contained in the firewall configuration files into platform-specific models (PSMs). Afterwards, each PSM is translated into a different network access-control PIM and an structural analysis to detect misconfigurations is performed. These PIMs are then aggregated into a global model, representing the access-control policy of the whole network. Operations are also performed over this global model to classify the information in “locally” or globally relevant.

5.2.1 Injection

The first step of our approach constitutes a mere translation between technical spaces where the textual information in the configuration files is expressed in terms of models. A PSM and a parser recognizing the grammar of each concrete firewall rule-filtering language present in the network system is required. In Listing 5.3 we excerpt a grammar for *CISCO PIX* and in 5.4 the corresponding one for Netfilter Iptables. In Section 5.5, we show how we use the to obtain the corresponding parser and PSM (The full definitions are available on the web of the project [7]). The integration of any other language will follow the same strategy.

Listing 5.3: Cisco grammar excerpt

```

Model:
    rules += Rule*;
Rule:
    AccessGroup | AccessList;
AccessGroup:
    'access-group' id=ID 'in' 'interface' interface=Interface;
Interface:
    id=ID;
AccessList:
    ('no')? 'access-list' id=ID
    decision=('deny' | 'permit')
    protocol=Protocol
    protocolObjectGroup=ProtocolObjectGroup
    serviceObjectGroup=ServiceObjectGroup
    networkObjectGroup=NetworkObjectGroup;
ProtocolObjectGroup:
    (pogId=ID)? sourceAddress=IPExpr sourceMask=MaskExpr;
ServiceObjectGroup:
    targetAddress=IPExpr targetMask=IPExpr;
NetworkObjectGroup:
    operator=Operator port=INT;
Operator:
    name=('eq' | 'lt' | 'gt');
Protocol:
    name= ('tcp' | 'udp' | 'ip');
IPExpr:
    INT '.' INT '.' INT '.' INT;

```

Listing 5.4: Netfilter Iptables grammar excerpt

```

Model:
    rules += Rule*;
Rule:
    declaration=ChainDeclaration | filter=FilterDeclaration;
FilterDeclaration:
    filter=FilteringSpec;
FilteringSpec:
    FilterSpec;
FilterSpec:
    'iptables' option=('-A' | '-D' | '-P')
    chain=Chain ((' -src' | '-s') ip=IPEXpr)?
    ('-i' interface=Interface)? ('-d' ipDst=IPEXpr)?
    ('-p' protocol=Protocol)? ('-m' matches=Protocol)?
    ('--sport' sourcePort=INT)? ('--dport' destinationPort=INT)?
    ('-j')? target=Target;
Interface:
    name=ID;
Protocol:
    Tcp | Udp | Icmp;
Tcp:
    'tcp';
Udp:
    'udp';
Icmp:
    'icmp';
Target:
    ID;
Chain:
    chainName = ID;
CustomChain:
    name=[ChainName];
ChainDeclaration:
    'iptables' '-N' ChainName;
ChainName:
    name=ID;
IPEXpr:
    ipByteExpr '.' ipByteExpr '.' ipByteExpr '.' ipByteExpr (IpRangeExpr)?;
ipByteExpr:
    INT;
IpRangeExpr:
    '/' INT;

```

Note that this step is performed without losing any information and that the obtained models remain at the same abstraction level as the configuration files.

5.2.2 Platform-specific to Platform-independent model

The second step of our approach implies transforming the PSMs obtained in the previous step to PIMs so that we get rid off the concrete details of the firewall technology, language and even writing style. Central to this step is the definition of a Network access-control metamodel able to represent the information contained in the PSMs. In the following we present and justify our proposal for such a metamodel.

Generally, firewall access-control policies can be seen as a set of individual

security rules of type $R_i : \{conditions\} \rightarrow \{decision\}$, where the subindex i specifies the ordering of the rule within the configuration file, *decision* can be accept or deny and *conditions* is a set of rule matching attributes like the source and destination addresses, the used protocol and the source and destination port.

Such a policy representation presents several disadvantages. First of all, the information is highly redundant and disperse, so that the details relevant to a given host or zone may appear, unassociated, in different places of the configuration file (potentially, containing up to several hundreds of rules). Metamodeling and model-driven technologies contain a big potential to reduce these issues, however, a proper representation must be chosen in order to maximize its benefits.

Second, this representation is not suited for representing the firewall policy in a natural and efficient way. Although firewall policies could be written by only using positive or negative logic (what leads however to over-complicated and not natural rule sets, impacting legibility and maintainability) a firewall access-control policy is better explained by expressing just rules in one sense (either negative or positive) and then exceptions (see [42] for a detailed study of the use of exceptions in access control policies) to the application of these rules. This way, in a close policy environment (where everything not explicitly accepted is forbidden) it is very common to define a security policy that accepts the traffic from a given zone and then denies it only for some elements of the zone. Native support for the representation of exceptions simplifies the representation and management of network policies while decreasing the risk of misconfiguration. The *custom chains mechanism*, recently provided by the linux iptables filter language, evidences the need for such a native support.

3.2.1. Network Access-control Metamodel.

The platform independent network access-control metamodel we propose here (see Figure 5.3) provides support for the representation of rules and exceptions. Moreover, our reverse engineering approach is designed to recover an exception-oriented representation of network security policies from configuration files disregarding if they use a good representation of exceptions like in the Iptables example in Section 5.1, or not, like in the Cisco PIX example in the same section.

Our metamodel proposal contains the following elements (note that, for simplicity, some attributes and references are not represented in the image):

- *Network Element*. Represents any subject (source of the access request) or object (target of the access request) within a network system. It is characterised by its ip address and its network mask.
- *Zone, Host, Server and Firewall*. Several different types of *Network Ele-*

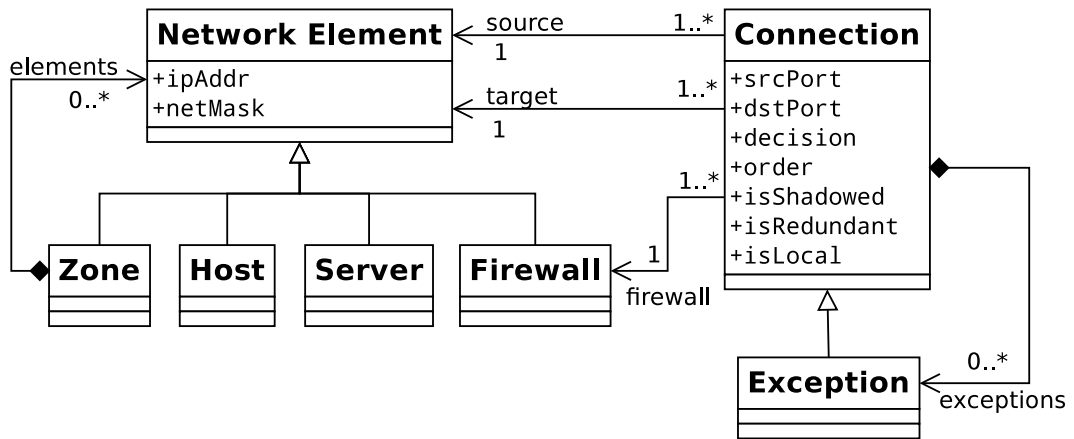


Figure 5.3: Filter PIM

ment may exist in a network environment. For the purpose of this paper, the relevant ones are: *Host*, *Zone* which in turn, contains other Network Elements, *Server* and *Firewall*. However, the list of elements can be extended to manage different scenarios, like the presence of routers, intrusion detections systems (IDSs), etc.

- *Connection*. Represents a connection between Network Elements. Apart from its source and target Network Elements, it is characterized by the following attributes: source and destination port, identifying the requested service; decision, stating if the connections is accepted or denied (our meta-model can represent open, close and mixed policies); order, reflecting the rule ordering in the corresponding configuration file; firewall, that identifies the firewall from where the connections were extracted; isLocal that tells is the connection is only locally relevant, isShadowed that identifies the connection as not reachable and finally, isRedundant, stating that the connection can be removed without affecting the policy.
- *Exception*. A connection may contain several exceptions to its application. These exceptions are connections with opposite decisions matching a subset of the elements matched by the containing connection.

3.2.2. PSM-to-PIM transformation.

Our PIM metamodel provides the means for representing network access-control policies in a concise and organised way. However, a proper processing of the information coming from the configuration files is required in order to fully exploit its capacities (a policy could be represented by using only *Connections* without using the *Exception* element). Therefore, the process of populating the PIM model from a PSM model is composed by two sub-steps.

The first sub-step fills our PIM with the information as it is normally found in configuration files, i.e., in the form of a set of rules representing exceptions with mixed positive and negative logic. Listings 5.5 and 5.6 show the ATL transformation performing this step for Iptables and Cisco PIX respectively.

Listing 5.5: Iptables PSM to PIM transformation

```

1 module IptablesPSM2PIM;
2 create OUT : NetworkAC from IN : IptablesDsl;
3
4 helper def : Rules : Sequence(IptablesDsl!FilterSpec) =
5   IptablesDsl!FilterSpec.allInstances();
6
7 helper def : AcceptRules : Sequence(IptablesDsl!FilterSpec) =
8   IptablesDsl!FilterSpec.allInstances()->select(e | e.chain.chainName = 'FORWARD'
9     ↪and e.target <> 'ACCEPT' and e.target <> 'RETURN' and e.target <> 'DROP');
10
11 helper def : ConcreteRules : Sequence(IptablesDsl!FilterSpec) =
12   IptablesDsl!FilterSpec.allInstances()->select(e | e.option = '-A');
13
14 helper context IptablesDsl!FilterSpec def : ruleOrder : Integer =
15   thisModule.Rules->indexOf(self);
16
17 helper def : UniqueIpAddr : Sequence(String) =
18   IptablesDsl!FilterSpec.allInstances().debug()->collect(e | e.ip)->select(e | not
19     ↪e.oclIsUndefined() and e <> '')
20   ->union(IptablesDsl!FilterSpec.allInstances()->collect(e | e.ipDst)->select(e |
21     ↪not e.oclIsUndefined() and e <> ''))
22   ->asSet()->asSequence().debug();
23
24 rule IptablesModel2NetworkACModel{
25   from
26     s: IptablesDsl!Model
27   to
28     t: NetworkAC!Model (
29       connections <- thisModule.AcceptRules,
30       elementKinds <- Sequence{firewallKind, serverKind, hostKind, zoneKind},
31       networkElements <- thisModule.UniqueIpAddr->collect(e | thisModule.
32         ↪createNetworkElement(e))->append(firewallElement)),
33     firewallElement: NetworkAC!NetworkElement (
34       kind <- firewallKind, name <- 'IptablesFirewall'),
35 }
36
37 rule ChainSpec2Network{
38   from
39     s:IptablesDsl!FilterSpec (s.chain.chainName = 'FORWARD' and
40     s.target <> 'ACCEPT' and s.target <> 'RETURN' and s.target <> 'DROP')
41   to
42     connection: NetworkAC!Connection (decision <- #Accept,
43     dstPort <- s.destinationPort.toString(),
44     firewall <- thisModule.resolveTemp(s.refImmediateComposite().
45     ↪refImmediateComposite().refImmediateComposite(), 'firewallElement'),
46     order <- s.ruleOrder, protocol <- s.protocol,
47     source <- thisModule.createNetworkElement(s.ip),
48     target <- thisModule.createNetworkElement(s.ipDst))
49 }
50
51 rule createConnectionFromChain{
52   from
53     chainInit:IptablesDsl!FilterSpec,

```

```

49     chainFollow:IptablesDsl!FilterSpec (chainInit.target = chainFollow.chain.
        ↪chainName and
50     chainFollow.ipDst.oclIsUndefined() and
51     chainFollow.target = 'RETURN')
52 to
53     connection:NetworkAC!Connection ->(NetworkAC!Model.allInstances()->first().
        ↪connections) (
54     decision <- #Deny,
55     dstPort <- chainInit.destinationPort.toString(),
56     firewall <- thisModule.resolveTemp(chainInit.refImmediateComposite().
        ↪refImmediateComposite().refImmediateComposite(), 'firewallElement'),
57     order <- chainFollow.ruleOrder + chainInit.ruleOrder,
58     protocol <- chainInit.protocol,
59     source <- thisModule.createNetworkElement(chainFollow.ip),
60     target <- thisModule.createNetworkElement(chainInit.ipDst)
61 }
62
63 rule createDestinationConnectionFromChain{
64 from
65     chainInit:IptablesDsl!FilterSpec,
66     chainFollow:IptablesDsl!FilterSpec (chainInit.target = chainFollow.chain.
        ↪chainName and
67     (not chainFollow.ipDst.oclIsUndefined()) and
68     chainFollow.target = 'RETURN')
69 to
70     connection:NetworkAC!Connection ->(NetworkAC!Model.allInstances()->first().
        ↪connections) (
71     decision <- #Deny,
72     dstPort <- chainInit.destinationPort.toString(),
73     firewall <- thisModule.resolveTemp(chainInit.refImmediateComposite().
        ↪refImmediateComposite().refImmediateComposite(), 'firewallElement'),
74     order <- chainFollow.ruleOrder + chainInit.ruleOrder,
75     protocol <- chainInit.protocol,
76     source <- thisModule.createNetworkElement(chainInit.ip),
77     target <- thisModule.createNetworkElement(chainFollow.ipDst)
78 }
79
80 unique lazy rule createNetworkElement{
81 from s: String
82 to t: NetworkAC!NetworkElement(ipAddr <- s, name <- s)
83 }

```

However, this representation can lead to policy anomalies and ambiguities. Concretely, as defined in [40], a firewall rule set may present the following anomalies:

Rule shadowing: a rule R is shadowed when it never applies because another rule with higher priority matches all the packets it may match.

Rule redundancy: a rule R is redundant when it is not shadowed and removing it from the rule set does not change the security policy.

Rule irrelevance: a rule R is irrelevant when it is meant to match packets that does not pass by a given firewall.

Listing 5.6: CiscoPIX PSM to PIM transformation

```

1 module CiscoPSM2PIM;
2 create OUT : NetworkAC from IN : CiscoDsl;
3
4 helper def : Rules : Sequence(CiscoDsl!AccessList) =
5   CiscoDsl!AccessList.allInstances();
6
7 helper context CiscoDsl!AccessList def : ruleOrder : Integer =
8   thisModule.Rules->indexOf(self);
9
10 helper def : UniqueIpAddr : Sequence(String) =
11   CiscoDsl!AccessList.allInstances()->collect(e | e.protocolObjectGroup.
12     ↪sourceAddress)->
13   union(CiscoDsl!AccessList.allInstances()->collect(e | e.serviceObjectGroup.
14     ↪targetAddress))->
15   asSet()->asSequence();
16
17 rule CiscoModel2NetworkACModel{
18   from
19     s: CiscoDsl!Model
20   to
21     t: NetworkAC!Model (
22       connections <- s.rules->select(e | e.oclIsKindOf(CiscoDsl!AccessList)),
23       elementKinds <- Sequence{firewallKind, serverKind, hostKind, zoneKind},
24       networkElements <- thisModule.UniqueIpAddr->collect(e | thisModule.
25         ↪createNetworkElement(e))->
26         append(firewallElement)),
27     firewallElement: NetworkAC!NetworkElement (
28       kind <- firewallKind,
29       name <- 'CiscoFirewall')
30 }
31
32 rule permitRule2Connection{
33   from
34     s: CiscoDsl!AccessList (s.decision = 'permit')
35   to
36     t: NetworkAC!Connection (
37       source <- thisModule.createNetworkElement(s.protocolObjectGroup.
38         ↪sourceAddress),
39       target <- thisModule.createNetworkElement(s.serviceObjectGroup.targetAddress
40         ↪),
41       dstPort <- s.networkObjectGroup.port.toString(),
42       order <- s.ruleOrder, decision <- #Accept, protocol <- s.protocol.name,
43       firewall <- thisModule.resolveTemp(s.refImmediateComposite(), '
44         ↪firewallElement'))
45 }
46
47 rule denyRule2Connection{
48   from
49     s: CiscoDsl!AccessList (s.decision = 'deny')
50   to
51     t: NetworkAC!Connection (
52       source <- thisModule.createNetworkElement(s.protocolObjectGroup.
53         ↪sourceAddress),
54       target <- thisModule.createNetworkElement(s.serviceObjectGroup.targetAddress
55         ↪),
56       dstPort <- s.networkObjectGroup.port.toString(),
57       order <- s.ruleOrder, decision <- #Deny, protocol <- s.protocol.name,
58       firewall <- thisModule.resolveTemp(s.refImmediateComposite(), '
59         ↪firewallElement'))
60 }
61
62 unique lazy rule createNetworkElement{
63   from s: String
64   to t: NetworkAC!NetworkElement(ipAddr <- s, name <- s)
65 }

```

Thus, the second sub-step, refines the initial PIM model and improve its internal organization to deal with the aforementioned problems. More specifically, this step applies the algorithm 1 on the PIM model (we describe the process for closed policies with exceptions, however, a version adapted to open policies would be straightforward):

1. Collect all the *Connection* elements C whose decision is *Accept*.
2. For each retrieved *Connection* C_i , get *Connections* C_j with the following constraints:
 - (a) C_j decision is *Deny*
 - (b) C_j conditions match a subset of the set matched by the conditions of C_i .
 - (c) C_j ordering number is lower than the C_i ordering number (if not, mark C_j as shadowed).

Then, for each retrieved C_j create an *Exception* element and aggregate it to the C_i . Remove the C_j *Connection*.
3. For each remaining *Connection* element C_j whose decision is *Deny* and is-Shadowed equals *false*:
 - mark C_j as isRedundant

Algorithm 1

```

1:  $C \leftarrow$  All Connections
2:  $C_{accept} \leftarrow C_i \in C (C_i.decision = Accept)$ 
3: for all  $C_i \in C_{accept}$  do
4:    $C_{deny} \leftarrow C_j \in C (C_j.decision = Deny \text{ and } Matched\ of\ C_j \subseteq\ matched\ C_i)$ 
5:   for all  $C_j \in C_{deny}$  do
6:     if  $C_j.order < C_i.order$  then
7:       Create Exception
8:       Remove  $C_j$ 
9:     else
10:       $C_j.IsShadowed \leftarrow true$ 
11:    end if
12:  end for
13: end for
14:  $C_{deny} \leftarrow C_j \in C (C_j.decision=Deny \text{ and } C_j.isShadowed=false)$ 
15: for all  $C_i \in C_{deny}$  do
16:    $C_j.IsRedundant \leftarrow true$ 
17: end for

```

The algorithm we have presented is a modification of the one presented in [41], e.g. to drop the requirement of using as input policy one free of shadowing and redundancy. On the contrary, it is meant to work on real configurations and helps to discover these anomalies: *shadowed* deny rules and *redundant* deny rules. The security expert can retrieve them easily from the PIM as any left *Connection* in the PIM with decision *Deny* is an anomaly and as such is marked as *isShadowed* or as *isRedundant*.

This algorithm can be complemented by a direct application of additional algorithms described in [40] to uncover other less important anomalies. Note that the correction of these anomalies will often require the segmentation and rewriting of the rules, therefore we consider the correction as an optional step to be manually triggered by the security expert after analysing the detected anomalies.

5.2.3 Aggregation of individual PIMs

At the end of the previous step we get a set of PIM's (one per firewall in the network). Clearly, an individual firewall gives only a partial vision of the security policy enforced in the whole network. In our example, analyzing one firewall will yield that the public host can access the SMTP server, however, this server can be also accessed by the private network with some exceptions. Thus, in order to obtain a complete representation of the network policy the individual PIM models have to be combined into one global network access-control model. Note that as we keep information regarding which firewall contains a given *Connection* element and the ordering with respect to the original configuration file, this step would be reversible, so that the individual policies may be reproduced from the global model.

Listing 5.7: PIM Aggregation transformation

```

1 module PIMAggregation;
2 create OUT : NetworkAC from IN : NetworkAC, IN1 : NetworkAC;
3
4 helper def : UniqueIpAddrs : Sequence(String) =
5   NetworkAC!NetworkElement.allInstances()
6   ->select(e | not e.ipAddr.oclIsUndefined())->collect(e | e.ipAddr)
7   ->asSet()->asSequence();
8
9 rule Model2AggregatedModel{
10  from
11    s1: NetworkAC!Model,
12    s2: NetworkAC!Model (s1 = NetworkAC!Model.allInstancesFrom('IN')->first() and
13      s2 = NetworkAC!Model.allInstancesFrom('IN1')->first())
14  to
15    t: NetworkAC!Model (
16      elementKinds <- s1.elementKinds,
17      connections <- s1.connections->append(s2.connections),
18      networkElements <- thisModule.UniqueIpAddrs->collect(e | thisModule.
19        ↪createNetworkElement(e))
20        ->append(NetworkAC!NetworkElement.allInstances()->select(e | e.kind->
21          ↪first().oclIsTypeOf(NetworkAC!Firewall))))

```

```

20 }
21
22 rule CopyConnections{
23   from
24     s: NetworkAC!Connection (s.decision = #Accept)
25   to
26     t: NetworkAC!Connection (decision ← s.decision, dstPort ← s.dstPort,
27     firewall ← s.firewall, order ← s.order,
28     protocol ← s.protocol,
29     source ← thisModule.createNetworkElement(s.source.ipAddr),
30     target ← thisModule.createNetworkElement(s.target.ipAddr),
31     exceptions ← s.exceptions )
32 }
33
34 rule CopyExceptions{
35   from
36     s: NetworkAC!Exception (s.decision = #Deny)
37   to
38     t: NetworkAC!Exception(decision ← s.decision, dstPort ← s.dstPort,
39     firewall ← s.firewall, order ← s.order, protocol ← s.protocol,
40     source ← thisModule.createNetworkElement(s.source.ipAddr),
41     target ← thisModule.createNetworkElement(s.target.ipAddr))
42 }
43
44 rule CopyFirewallElement{
45   from
46     s: NetworkAC!NetworkElement (s.kind->first().oclIsTypeOf(NetworkAC!Firewall))
47   to
48     t: NetworkAC!NetworkElement (
49     kind ← NetworkAC!Firewall.allInstancesFrom('IN')->first(), name ← s.name)
50 }
51
52 unique lazy rule createNetworkElement{
53   from
54     s: String
55   to
56     t: NetworkAC!NetworkElement(ipAddr ← s, name ← s)
57 }

```

We obtain the global model by performing a model union operation between the individual models, so that no *Network Element* or *Connection* is duplicated. Listing 5.7 shows an ATL transformation performing the aggregation of two PIM models. As an extra step, a refining transformation is performed to assign the proper type to the *Network Elements*. This step is performed by analysing the *ip* addresses and the incoming and outgoing connections. This way, we are able to establish if a network element is a zone or being an individual network element behaves as a host or a server (a unique firewall element is created upon the initialization of each PIM model in order to represent the firewall the rules come from). Once we have obtained the global model, some operations become available.

First of all, local *Exceptions* and *Connections*, i.e., *Exceptions* and *Connections* that only make sense in the context of a concrete firewall, can be identified (so that they can be filtered out when representing the global policy.). Local exceptions are usually added due to the mechanisms used to enforce the global policy. As an exam-

ple, in the Listing 5.9 the elements in the network zone 222.111.0.0 are not allowed to send or receive smtp messages. However, elements in 222.111.2.0 are allowed to send them regarding the configuration Listing 5.2. This contradiction is due to the enforcing architecture that imposes the traffic to pass through a certain firewall (in this case, hosts in the local network are meant to access the DMZ through the second firewall). Algorithm 2 detects local *Exceptions* and *Connections* and it works as follows:

1. Collect all the *Exceptions* E in the aggregated model.
2. For each *Exception* E_i , L is a set of *Connections* C with the following constraints:
 - (a) C_i is retrieved from a firewall different that the one containing E_i
 - (b) C_i conditions, are subset (or equal) of E_i conditions.

If the obtained set of *Connections* L is not empty:

- Mark E_i as local.
- For each C_i in L , mark C_i as local if it has not been already marked.

Algorithm 2

```

1:  $C \leftarrow$  All Connections
2:  $E \leftarrow$  All Exceptions
3: for all  $E_i \in E$  do
4:    $L \leftarrow C_i \in C$  ( $C_i$ .firewall  $\neq$   $E_i$ .firewall and Matched of  $C_i \subseteq$  matched  $E_i$ )
5:   if  $L \neq \emptyset$  then
6:      $E_i$ .IsLocal  $\leftarrow$  true
7:     for all  $C_i \in L$  do
8:        $C_i$ .IsLocal  $\leftarrow$  true
9:     end for
10:  end if
11: end for

```

This will be also useful when extracting a representation of the network topology covered by the firewalls (see next section).

5.3 Applications

Once all the access-control information is aggregated in our final PIM, we are able to use the model in several interesting security application scenarios.

Metrics and advanced queries. First of all, having the access-control information of a network represented as a model, enables the reutilization of a plethora of well-known, off-the-shelf MDE tools. Editor generators, model transformation engines, etc. become automatically available. An immediate application would be the use of the well-known OCL query language to calculate interesting metrics on the model and perform some advanced queries on the rules represented in it. In the following example, we query our model (in the example, the context of *self* is the root of our PIM) for the existence of any connection allowing the administrator host (111.222.2.54) to connect to the server (111.222.1.17):

```
Evaluating:

self.connections->exists(e | e.source.ipAddr='111.222.2.54' and
                        e.target.ipAddr='111.222.1.17')

Results:
false
```

Forward engineering. Our PIM model extracts and abstracts information from working networks. Nevertheless, the PIM is still rich enough to be able to be used as starting point for the regeneration of the configuration files if necessary (e.g. after modifications on the PIM to update the security policy of the network according to the new requirements). In that sense, some existing forward engineering efforts[81, 16] that produce firewall configurations from high level representations can be reused.

Visualization of the topology. Our PIM can also be used to derive the topology of the network, i.e., the arrangement of components and how the information flow through them. For this purpose, a model-driven visualization tool like Portolan¹ can be used. A transformation from our aggregated PIM towards the Portolan Cartography model (Portolan is able to graphically represent any model corresponding to its Cartography metamodel) has been written. This transformation analyzes the global PIM to first, extract the *Firewall* elements and represent them as nodes. Then, represent the other *Network Elements* also as nodes and the local containment of *Zones*. Finally, it extracts the *Connections* and build the links between each *Connection* source *Network Element* to the corresponding *Firewall* element and from the *Firewall* element to the target *Network Element*.

In Figure 5.4 we show the visualization the tool provided. In the figure, servers (element 111.222.1.17), firewalls, zones and contained elements are easily identifiable as well as the enabled connections between them. If we compare this figure

1. <http://code.google.com/a/eclipselabs.org/p/portolan/>

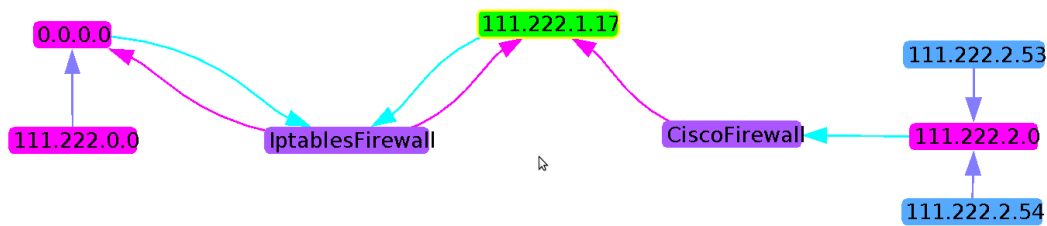


Figure 5.4: Extracted network topology

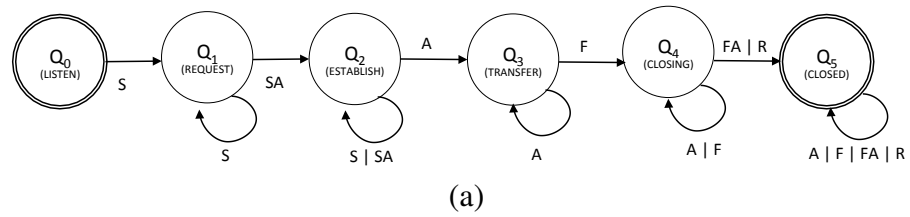
with the figure 5.1 presented in section 5.1, we can see that the topology is accurately represented.

5.4 Analysis of stateful misconfigurations

Firewall configurations are evolving into dynamic policies that depend on protocol states. In order to show the flexibility of our approach, in this section we extend it to the representation and the analysis of stateful firewalls.

The main goal of a firewall is to control network traffic flowing across different areas of a given local network. It must provide either hardware or software means to block unwanted traffic, or to re-route packets towards other components for further analysis. In the stateless case, the filtering actions, such as accepting or rejecting packet flows, are taken according to a set of static configuration rules. These rules only pay attention to information contained in the packet itself, such as network addresses (source and destination), ports and protocol. The main advantage of stateless firewalls is their filtering operations speed. However, since they do not keep track of state connection data, they fail at handling some vulnerabilities that benefit from the position of a packet within existing streams of traffic. Stateful firewalls solve this problem and improve packet filtering by keeping track of connection status. Indeed, they can block those packets that are not meeting the valid state machine of a given connection-oriented protocol. As with stateless packet filtering, stateful filtering intercepts the packets at the network layer and verifies if they match previously defined security rules. Moreover, stateful firewalls keep track of each connection in an internal state table.

Stateful firewalls provide fine-grained filtering capabilities to protect networks against complex attacks. For instance, stateful filtering can be used in order to detect and drop anomalous behavior in TCP traffic (or for any other connection-oriented protocol) flows that progress via invalid connection states. Automata can be used to describe the progression of states based on header flags. This way, illicit scanning activities [67] or brute force termination attacks [12] can be described in the



Event label	Flag set	Event description
S	{SYN}	Request to open connection
SA	{SYN+ACK}	Agree to open connection
A	{ACK}	Acknowledgement
F	{FIN}	Request to close connection
FA	{FIN+ACK}	Close connection gracefully
R	{RST}	Tear down connection
I	Set of all other invalid flag combinations	

(b)

State \ Event	S	SA	A	F	FA	R	I
Q ₀ (LISTEN)	Q ₁	∅	∅	∅	∅	∅	∅
Q ₁ (REQUEST)	Q ₁	Q ₂	Q ₁	∅	∅	∅	∅
Q ₂ (ESTABLISH)	Q ₂	Q ₂	Q ₃	∅	∅	∅	∅
Q ₃ (TRANSFER)	∅	∅	Q ₃	Q ₄	∅	∅	∅
Q ₄ (CLOSING)	∅	∅	Q ₄	Q ₄	Q ₅	Q ₅	∅
Q ₅ (CLOSED)	∅	∅	Q ₅	Q ₅	Q ₅	Q ₅	∅
∅ (INVALID)	∅	∅	∅	∅	∅	∅	∅

(c)

Figure 5.5: Progression of a TCP connection exhibiting normal behavior, based on [95]. (a) Simplified TCP automaton (it represents together the two separate automata, one for the client and one for the server, of the traditional TCP finite state machine). (b) Events description. (b) Transition table.

configuration of the stateful firewall, so that albeit of being dropped, such activities are also reported to the security officer. Assume the simplified automaton in Figure 5.5(a), in which we describe the progression of a TCP connection exhibiting normal behaviour [95]. As represented by this automaton, a TCP connection progresses from state to state based on the information contained in the headers of the TCP packets exchanged between two peers, and specified as the TCP traffic flag combination in Figure 5.5(a). Based on this approach, the security officer can now signal invalid transitions, as represented in Figure 5.5(b) by the symbol \emptyset . This invalid state corresponds to anomalous TCP transitions. Illicit scanning activities and brute force termination attacks can easily be identified by means of these anomalous transitions. Existing tools such as *QueSO* [2], *NMAP* [1] and *Trafscrambler* [4] are available on-line in order to conduct such kind of illicit activities. Using the information in Figure 5.5(b), the security officer can define a list of stateful access control rules for a specific stateful firewall, e.g., a list of stateful rules for a firewall based on the Linux operating system.

Netfilter Iptables [3], provides stateful filtering capabilities in order to grant access to network traffic based on already existing connections. As the implementation of stateful capabilities differ from one vendor to another, hereinafter, we will focus in Netfilter Iptables stateful implementations. Notice however that all the operations we present here are implemented at PIM level, so they are completely reusable for any other firewall vendor offering similar capabilities.

This stateful feature of Netfilter is based on the use of the *iptables* and *conntrack* modules. The *iptables* module defines tables (e.g., filtering tables) and chains (e.g., input, forward and output chains) to conduct actions over packets traversing the network stack. The *conntrack* module (short for *connection-tracking*) provides to Netfilter with the ability for maintaining state information about the packets the firewall is examining. Therefore, the combination of both modules allows the user to define stateful filtering rules for connection-oriented protocols.

Suppose the addition of the following rules in the tables of a Netfilter firewall controlling Internet connections directed towards a network server, and whose default policy is open (i.e., it accepts all network packets not matching any given rule):

Listing 5.8: Firewall stateful configuration

```
01: iptables -P FORWARD ACCEPT
02: iptables -N InvRQ
03: iptables -A FORWARD -p tcp -d 5.6.7.8 --m conntrack --ctstate NEW !
--tcp-flags ALL SYN --ctdir REPLY -j InvRQ
04: iptables -A InvRQ -j LOG --log-prefix 'InvRQ'
05: iptables -A InvRQ -j DROP
```

In the previous example, the main action (cf. line 03) is based on the *conntrack match* for *iptables*, which makes it possible to define filtering rules in a much more granular way than simply using stateless rules or rules based on the *state* match (cf. reference [3] and citations thereof a more extensive description). This is defined by providing the parameter *m conntrack* to the rules. The *ctstate NEW* parameter is used to instruct the firewall to match those TCP packets in the *conntrack* table that are seen for first time. The *syn* parameter, preceded by the '!' symbol, is used to exclude from such packets, those with the SYN flag activated. Finally, the *ctdir ORIGINAL* parameter is used to exclude those packets flowing from the server to the Internet. As a result, the above rules allow to report and drop those TCP traffic connections across the *forward* chain that exhibit the invalid behavior defined in the first row in Figure 5.5(b), i.e., transitions from the initial state to the invalid one, as a result of invalid flag combinations. In other words, it corresponds to the discovery and prohibition of TCP traffic connections that may be associated to illicit scanning activities.

In order to address the invalid states in the second and third rows in Figure 5.5(b), we can complement now the previous set of rules with the following ones:

Listing 5.9: Firewall stateful configuration

```
06: iptables -N InvRP
07: iptables -A FORWARD -p tcp -s 5.6.7.8 --m conntrack --ctstate ESTABLISHED
--ctdir ORIGINAL --ctstatus SEEN REPLY -j InvRP
08: iptables -A FORWARD -p tcp -d 5.6.7.8 --m conntrack --ctstate ESTABLISHED
--ctdir REPLY --ctstatus ASSURED -j InvRP
09: iptables -A InvRP --tcp-flags ALL SYN -j RETURN
10: iptables -A InvRP --tcp-flags ALL SYN,ACK -j RETURN
11: iptables -A InvRP --tcp-flags ALL ACK -j RETURN
```

```
12: iptables -A InvRP -j LOG --log-prefix 'InvRP'  
13: iptables -A InvRP -j DROP
```

Equivalent reasoning can be used to complement the set of rules and cover all the remainder cases of anomalous transitions in Figure 5.5(b). The result shall be a complete set of stateful rules covering each of the invalid transitions. Notice that the gain in expressivity that the stateful feature offers to a security officer leads to error-prone configurations. For instance, in the case of open policies (as assumed in the above examples), it is possible to end up with incomplete rule sets, so that some threats are not appropriately covered. Therefore, leading to a weak enforcement of policies. This can be the case in which some parameters like *ctstatus*, *ctdir*, *tcp-flags*, etc., are not appropriately used. For instance, the omission (by mistake) of the '!' symbol in line 03 of our previous example would lead to a situation in which all the anomalous transitions of the first row in Figure 5.5(b) are now allowed by the firewall, i.e., the firewall allows all those illicit scanning activities associated to these transitions. In the case of closed policies (i.e., the firewall blocks all network packets not matching any given rule) potential misconnection in the inner logic of the protocol can lead to services not responding as expected (e.g., the firewall is instructed to accept the initial phase of the connection establishment, but blocks the remainder phases). Again, the omission of the '!' symbol in line 03 would instruct the firewall to drop the initial connection establishment to the server, and therefore blocking all potential communication with it. In the general case, in which there is a combination of open and closed default policies, the definition of rules can definitively be much more complex than in the normal stateless case. Indeed, it is required a deep knowledge of the inner logic of the states of the protocols being dealt with. Otherwise, flawed configurations are likely to happen. In the sequel, we address this problem and provide an automatic solution to handle it.

5.4.1 Metamodel and Parser extension

To make it able to deal with stateful information as well, the network access-control metamodel and the Iptables parser needed to be extended.

Regarding the metamodel, State and Event entities have been added. These two fields correspond to the Transition attribute presented in previous sections, for our generic stateful filter model. This way, State represents the state a connection is w.r.t. the finite state machine of a connection-oriented protocol (e.g., TCP, TCP, DCCP, ATM, Frame Relay, TIPC, SCTP, etc.); and Event represents the triggering condition for the transitions of such a state machine.

As for the parser, the special *conntrack* parameters of iptables rules were added to the grammar, so that the XTEXT framework produces the corresponding Iptables stateful metamodel and injector. In Listing 5.10 we show the required changes for

Listing 5.10: Netfilter Iptables grammar extension for Stateful

```

FilterSpec:
  'iptables' option=('A' | '-D' | '-P')
  chain=Chain ('-s' ip=IPEXpr)?
  ('-i' interface=Interface)? ('-d' ipDst=IPEXpr)?
  ('-p' protocol=Protocol)?
  ('--sport' sourcePort=INT)?
  ('--dport' destinationPort=INT)? (neg?='!')?
  (syn?='--syn')? ('--m' matches=Match)?
  ('--ctstate' states+=State (',' states+=State)*)?
  ('--ctdir' dir=Dir)?
  ('--ctstatus' status=Status)?
  ('--state' states+=State (',' states+=State)*)?
  ('--tcp-flags' examFlags+=TCPFlag
  (',' examFlags+=TCPFlag)* flags+=TCPFlag
  (',' flags+=TCPFlag)*)?
  ('-j')? target=Target
  ('--log-prefix' lp=LP)?
;

Match:
  name=(Conntrack|StateMatch)
;

TCPFlag:
  name=(Syn | Ack | Fin | Rst | All | None)
;

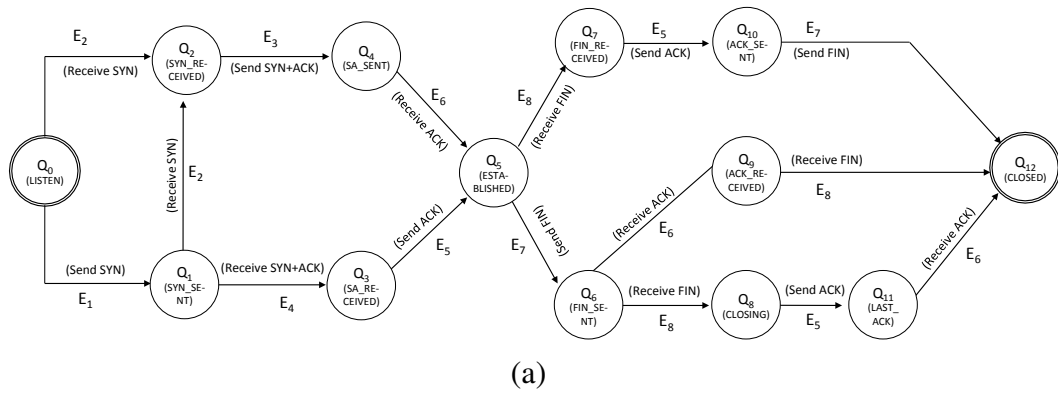
```

the grammar.

5.4.2 Intra-state rule misconfiguration

In [37], the authors uncovered a new type of misconfiguration, denoted as intra-state rule misconfiguration. Intra-state rule misconfiguration leads to flawed configurations in which some stateful actions, according to a connection-oriented protocol, are conducted by the firewall, while other related actions are not. They provided an algorithmic solution to discover and correct explicit conflicting rules, so that the resulting set gets consistent to the action of those rules with higher priority in order. In this section, we complement the algorithmic solutions in [37], in order to assist and guide in the correction of the more complex case, in which intra-state misconfiguration is driven by the omission of explicit rules in the set. The principle of our approach is based on the specification of general automata. Such automata describe the different states that traffic packages can take throughout the filtering process.

Algorithm `audit_rule_set` complements the previous intra-state rule misconfiguration management process presented in [37]. Its pseudocode is summarised in Algorithm 1. It uses as input a stateful rule set R , in which each rule specifies an Action (e.g., ACCEPT or DENY) that applies to a set of condition attributes, such as SrcAddr, DstAddr, SPort, DPort, Transition, and Protocol. The Protocol attribute corresponds to a connection-oriented protocol. An automaton A characterising the progression of a connection for such a protocol is also provided to the algorithm. Finally, the identifiers for the initial (Q_0), final (Q_n) and invalid (B) states are



Event label	Flag set	Direction	Event description
E ₁	{SYN}	Sender path	Send SYN
E ₂	{SYN}	Receiver path	Receive SYN
E ₃	{SYN+ACK}	Sender path	Send SYN+ACK
E ₄	{SYN+ACK}	Receiver path	Receive SYN+ACK
E ₅	{ACK}	Sender path	Send ACK
E ₆	{ACK}	Receiver path	Receive ACK
E ₇	{FIN}	Sender path	Send FIN
E ₈	{FIN}	Receiver path	Receive FIN
E ₉	Set of all other invalid flag combinations (Sender path)		
E ₁₀	Set of all other invalid flag combinations (Receiver path)		

(b)

Figure 5.6: (a) Automaton of a given protocol P (for simplicity, the invalid state is not shown). (b) Events description for the automaton.

also used as input parameters. The main steps of the algorithm are:

- Build a set S containing all possible paths of valid transitions connecting the initial (Q₀) and the final (Q_n) states of automaton A (Line 6);
- Build a set T containing all the transitions of automaton A to reach the invalid (B) state (Line 7);
- Verify the coverage of either S or T (Lines 8e20), w.r.t. the Action (i.e., ACCEPT or DENY) and Transition attributes of all the rules in R;

To build S, the algorithm uses function `find_all_paths`. Function `find_all_paths` recursively performs exhaustive search of automaton A and keeps track of all the possible paths of valid transitions. Similarly, T is built using function `transitions_to_state`. This function starts at B (invalid state) and recursively builds T with all the transi-

tions it finds over a one-dimensional vector. function `cover_with_rules` provides a mapping between the sets of transitions in L , and those in rules of R that are consistent with the attributes of r_1 . `extract_missing_rules` simply pulls out from C one series of missing rules.

Algorithm 3 `audit_rule_set(A, R, Q0, Qn, ⊘)`

Input: A , protocol automaton

Input: R , stateful rule set

Input: Q_0 , initial state of automaton A

Input: Q_n , final state of automaton A

Input: \ominus , invalid state of automaton A

$S \leftarrow \text{find-all-paths}(A, Q_0, Q_n)$;

/ S : set of sets, s.t. every element E in S is a set of transitions (i.e., a path) connecting Q_0 and Q_n */*

$T \leftarrow \text{transitions-to-state}(A, \ominus)$;

/ T : set of transitions, s.t. every t in T is a terminal transition in A to reach \ominus */*

repeat

$r_i \leftarrow \text{next-unvisited-rule}(R)$;

if ($r_i[\text{Action}] = \text{ACCEPT}$) **then**

$L \leftarrow \text{prune-paths}(S, r_i[\text{Transition}])$;

/ $L \subseteq S$, such that every path P in L is a set of transitions connecting Q_0 and Q_n via $r_i[\text{Transition}]$ */*

else if ($r_i[\text{Action}] = \text{DENY}$) **then**

$L \leftarrow T$;

end if

$C \leftarrow \text{cover-with-rules}(L, R, r_i)$;

$M \leftarrow \text{extract-missing-rules}(C)$;

/ M : set of mutually disjoint rules derived from R , s.t. $M \cap R = \emptyset$; let m be a rule in M , then $m[\text{Action}] = r_i[\text{Action}]$, Address and Port attributes of m are consistent to rules in R , and $m[\text{Transition}]$ is necessary to fully cover the set of transitions in L */*

if ($M \neq \emptyset$) **then**

report('Anomaly in R due to r_i ');

report('Update R based on rules in M ');

end if

until `no-more-rules-to-visit`(R);

/ `no-more-rules-to-visit`(R) holds true whenever all rules in R but the last are reported as visited (the last rule is not inspected given that it is just a mark to the default policy) */*

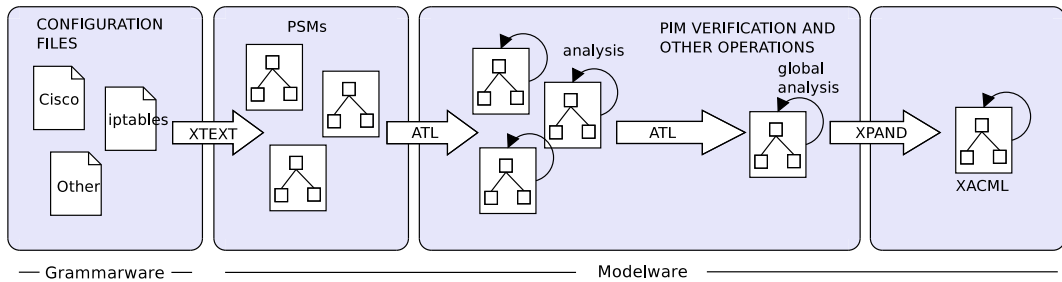


Figure 5.7: Extraction approach implementation

5.5 Tool Support

In order to validate the feasibility of our approach, a prototype tool[7], able to work with two popular firewall filtering languages *Linux Iptables* and *Cisco PIX*, has been developed under the Eclipse² environment. Figure 5.7 summarizes the steps and technological choices we made for the prototype development.

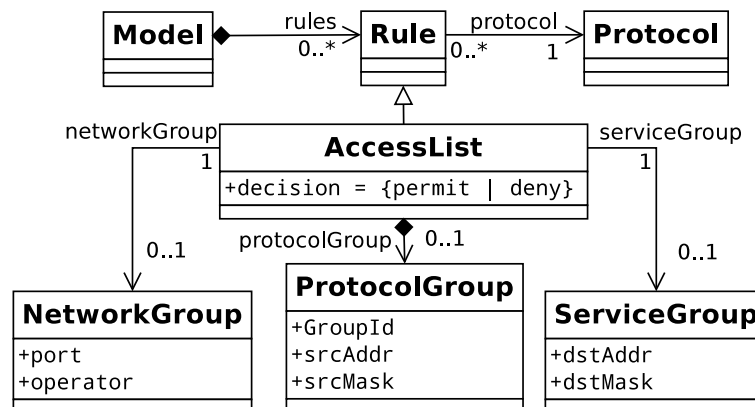


Figure 5.8: Cisco Metamodel

The tool implements the first step of our approach (the injection process) with Xtext³, an Eclipse plugin for building domain specific languages. As an input to this tool we have written simple yet usable grammars for the two languages supported by our tool. By providing these two grammars the Xtext tool creates for us the corresponding metamodels depicted in Figures 5.8 and 5.9 along with the parser and the injector needed to get models out of the configurations files.

2. <http://www.eclipse.org/>

3. <http://www.eclipse.org/Xtext/>

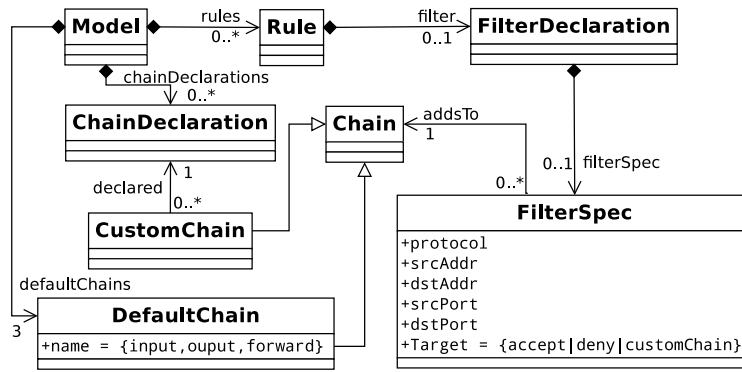


Figure 5.9: Iptables Metamodel

The transformations from the PSMs to the PIM along with the detection of anomalies have been written using the model transformation language ATL[51], both in its normal and in-place (for model refining[93]) modes. Same for the PIM’s aggregation process.

As for the stateful analysis, Figure 5.10 summarises the approach followed by our prototype to handle misconfigured files.

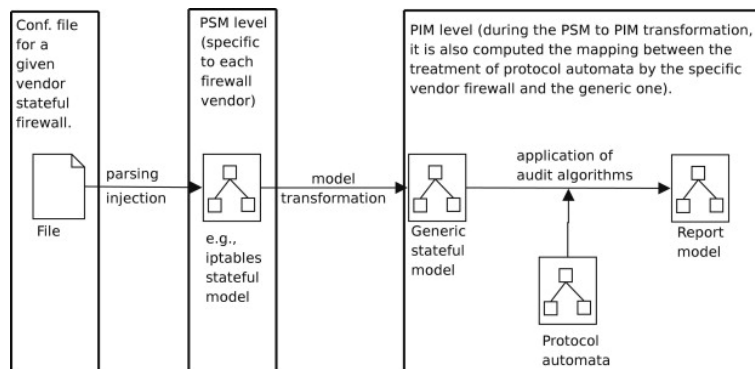


Figure 5.10: Stateful analysis approach

It comprises the following steps.

- (1) Parsing and injection of already deployed configuration files into models at the PSM level;
- (2) Transformation of models from the PSM level to the PIM level;
- (3) Alignment of models at the PIM level w.r.t. a given protocol automaton;
- (4) Execution of misconfiguration algorithms and reporting of the discovered anomalies;
- (5) Generation of corrective rules.

The implementation of Steps 1 and 2 is similar to those of the stateless case. However, during step 2, dealing with the transformation of rules from the PSM level to the PIM level, we also compute step 3, the mapping between the treatment

of protocol automata by the specific vendor firewall and the generic one (notice that the definition of the automata is also performed at the model level as an EMF model for representing automata have been also developed). We show an excerpt of this transformation in Listing 5.11 where the rule *filterNew2ConnectionsQ0* transforms Iptables rules towards *Connections* of our PIM and properly sets the *State* and *Event* w.r.t. to the automata in Figure 5.6.

Listing 5.11: Stateful PSM to PIM transformation

```

1 rule filterNew2ConnectionsQ0{
2   from
3     s1: Iptables!FilterSpec,
4     s2: Iptables!FilterSpec (thisModule.FirstPathRules->includes(s1)
5       and s1.isOfState('NEW', 'SYN')
6       and s2.isOfState('NEW', 'SYN')
7       and s2.ip = s1.ipDst
8       and s2.ipDst = s1.ip)
9   to
10    t: StatefulPIM!Connection -> (StatefulPIM!Network.allInstancesFrom('OUT')->
11      ↪first().connections) (
12      srcPort <- s1.sourcePort,
13      dstPort <- s1.destinationPort,
14      kind <- if s1.target = 'ACCEPT' then #ACCEPT else #DENY endif,
15      protocol <- s1.protocol,
16      states <- state,
17      setFlags <- event,
18      desHost <- thisModule.createHost(thisModule.findHost(s1.ipDst)),
19      srcHost <- thisModule.createHost(thisModule.findHost(s1.ip))
20    ),
21    state: StatefulPIM!State (
22      name <- 'Q0'
23    ),
24    event: StatefulPIM!Event (
25      name <- 'E2'
26    ),
27    t2: StatefulPIM!Connection -> (StatefulPIM!Network.allInstancesFrom('OUT')->
28      ↪first().connections) (
29      srcPort <- s2.sourcePort,
30      dstPort <- s2.destinationPort,
31      kind <- if s2.target = 'ACCEPT' then #ACCEPT else #DENY endif,
32      protocol <- s2.protocol,
33      states <- state2,
34      setFlags <- event2,
35      desHost <- thisModule.createHost(thisModule.findHost(s2.ipDst)),
36      srcHost <- thisModule.createHost(thisModule.findHost(s2.ip))
37    ),
38    state2: StatefulPIM!State (
39      name <- 'Q0'
40    ),
41    event2: StatefulPIM!Event (
42      name <- 'E1'
43    )
44  }

```

The application of the audit algorithms presented in is done at the PIM level. The algorithms themselves have been encoded as ATL transformations. This way, the application of the algorithms and functions is independent of the specific fire-

wall. The output of the audit process is a new model that contains all the necessary feedback to handle the detected misconfiguration, such as the missing rules needed to handle it.

The visualization of the topology relies on Portolan, and the transformations between our PIM and the Portolan model on ATL.

Extracting AC Models from Relational Databases

Relational databases are at the core of most companies information systems, hosting critical information for the day to day operation of the company. Securing this information is therefore a critical concern. For this purpose, both, researchers and tool vendors have proposed and developed security mechanisms to ensure the data within the database system is safe from possible threats. Due to its relative conceptual simplicity, one of the most used mechanisms within DataBase Management Systems (DBMSs) are *access control (AC)* policies where Role-based access control (RBAC) [85] is the current trend.

However, and despite the few methods that attempt to automatically derive these policy implementations from high-level security specifications[70, 17, 15], the task of implementing an access control security policy remains in the vast majority of cases a manual process which is time-consuming and error-prone. Besides, several database mechanisms may be needed in the implementation of the policy, e.g., triggers can be used to add fine-grained control on privileges, scattering the policy and increasing the complexity of the definition process. Furthermore, as security requirements are rarely static (new application scenarios, new users, etc), frequent modifications of the security policy implementation are required, what increases the chances of introducing new errors and inconsistencies.

In this context, discovering and understanding which security policies are actually being enforced by the database system comes out as a critical necessity. This is a necessary condition for the reengineering of the current policies to adapt them to evolving needs of the company and also to detect inconsistencies between the

enforced and the desired policies. The main challenge for this discovery process is bridging the gap between the vendor-dependent policy representation and a more logical model that 1 - express these policies in a way that abstracts them from the specificities of particular database systems and 2 - that can be understood by security experts with no deep knowledge of the particularities of each vendor. Representing and processing the security policies at this logical level is much easier than a direct exploration of the database dictionary where the security information is scattered among different dictionary tables whose structure is unfortunately not standardised. Additionally, this logical model would also allow us to implement all analysis/evolution/refactoring/manipulation operations on the security policies in a vendor-independent and reusable way.

The goal of this chapter is then two-fold. First we provide a means to represent such logical models for security concerns in relational database systems. Secondly, we describe a reverse engineering approach that can automatically create this logical model out of an existing database. Reverse engineering, as a process aimed to represent running systems at higher abstraction level, has been proved useful in many domains [28], including database systems [62], [33]. However, these works have focused on the database structure and ignored the security aspects. We intend to cover this gap.

Additionally, we discuss possible applications and benefits of using a model-based representation given the fact that this enables to reuse in the security domain the large number of model-driven techniques and tools. Model manipulation techniques can then be applied to visualize, analyze, evolve, etc the model. Then, a forward engineering process could be launched in order to generate the new security policy implementation ready to import in the target database system.

6.1 Motivation

Security in databases heavily relies on the implementation of access control mechanisms that ensure that only authorised users have access to read/modify a given piece of data.

Access control manages the assignment of privileges or permissions, i.e., the execution of operations, on system objects to subjects. It has been largely studied by the research community and adopted by database tool vendors [20]. Mandatory Access Control (MAC), Discretionary Access Control(DAC) and Role-based Access Control (RBAC) are the most successful models. RBAC (with DAC capabilities) is currently the most popular one and will be the focus of this chapter.

Such a RBAC-like access control security policy is enforced in a database system by using a set of different mechanisms. Moreover, the information of the im-

plemented policies is scattered around several tables and columns in the internal database dictionary, making it very difficult to quickly grasp the security constraints of a database. This section provides an introduction to these security mechanisms while next one presents the security model as a way to provide a more homogeneous representation of all of them.

Privileges: We can divide the privileges that can be granted in a DBMSs in five categories: *Database-level* privileges, for those privileges that imply creation of database objects including users and roles. *Table-level* privileges for those that implies table, columns and index access. *Permission-delegation* privileges to delegate permission administration to users and roles. *Execute* privileges for the executable elements (stored procedures and functions) and *Session* privileges. The corresponding privileges for each category are listed below:

- The table level privileges are the following: SELECT, UPDATE, INSERT and DELETE.
- The database level privileges include CREATE, ALTER and DROP.
- GRANT and REVOKE are the privileges in the permission-delegation category. They can be granted to users or roles so that the permission administration is delegated.
- The EXECUTE privilege is the only one in the execute category. It is meant to be granted on procedural code elements.
- SET and CONNECT are the privileges in the session category. They manage the ability of a user to access a database and to act as a given role.

The list of privileges that exist within concrete DBMSs is longer than the one presented here (e.g., Oracle defines more than one hundred privileges to be granted). However, they are either vendor specific or are meant to be used only in very specific cases.

Creating and assigning roles. A basic role-based access control policy can be set by using standard SQL commands [49] to create and assign roles (and permissions), although not all commercial DBMSs fully support the SQL standard (e.g., MySQL does not give support to roles so privileges are directly assigned to users). In the following the relevant SQL commands are showed and explained.

- CREATE / DROP for roles: Those commands enable the creation and deletion of roles from the database.

```
1 //Creates the role test_role in the database
2 CREATE ROLE test_role
```



```

3 //Removes the role test_role form the database
4 DROP ROLE test_role

```

- **GRANT / REVOKE:** The action of these commands is twofold, as they can be used on privileges but also on roles. When used on privileges they allow to grant/revoke a given privilege to a user or a role on a specific database object. When used on roles, it allows to grant/revoke a role to a user or to another role in DBMSs that supports role hierarchies.

```

1
2 //Grants select, insert update and delete privileges on test_table to
3 //the test_role role
4 GRANT SELECT, INSERT, UPDATE, DELETE ON test_table TO test_role;
5 //Grants the test_role role to test_user
6 GRANT USER test_user test_role;
7
8 //Revoke the delete privilege from the test_role
9 REVOKE DELETE ON test_table FROM test_role;
10 //Revoke the role test_role form the user test_user
11 REVOKE test_role FROM test_user;

```

- **SET:** This command allows the user to choose which role or roles between the roles she has been granted will be used in the current session. Depending of the concrete implementation and vendor constraints a dynamic SoD could be enforced so that a user can not hold at the same time two incompatible roles.

```

1 //Set the role test_role as the active role for the user
2 //in the current session
3 SET ROLE test_role;

```

Some of the explained commands admit the use of modifiers that enhance their behaviour and their semantics (e.g., granting a role with the `GRANT OPTION` allows the grantee to, in turn, grant the role to other users). Again, as the basic commands, they are supported only by certain vendors and just until some extent.

View definitions. Views can be used as a fine-grained access control mechanism. A view filters table rows and columns with respect to a desired criteria so when a subject is granted privileges over a view, is actually being granted privileges over a table or a set of tables but with certain constraints. This actually represents *column-level* and *content-based (row level)* access control.

Functions, stored procedures and triggers. Stored procedures, triggers and functions may also participate in the implementation of an access control policy.

An stored procedure can be executed with two different sets of permissions, definer's rights or invoker's rights. If it is executed with definer's right this, in fact, implies the invoker obtains transitive rights on the objects used within the code of the stored procedure. This constitutes a security feature in several DB system

as it allows to give very few permissions to application roles and to encapsulate functionality in the procedures. However, it is important to be able to "see" this information as it could be a security threat (some user not supposed to insert in a given table may do it by transitivity). Moreover, if the definer's set of rights is important, potentially including administration permissions, a user or role winning permission on the modification of the procedure could modify it to gain access to critical information. Making this information easy to grasp, will allow for a better organization of the policy.

Triggers can also be used to set constraints on granted privileges. This way, generic privileges granted to users will be further filtered depending on the context of the access request. Typically, triggers will be defined to monitor the modification actions on certain tables. Once the trigger is fired, a condition is evaluated to decide whether to go ahead or not with the action based on contextual information (e.g. time of the day, values of the modification action,...). As part of the condition a stored procedure could be called.

Some vendors offer even more sophisticated mechanisms, e.g., Oracle Virtual Private Databases (VPD)[75], which uses PL/SQL to associate an additional where clause with SELECT expressions depending on the user/role who launches the SELECT.

6.2 RDBMS Access-Control Metamodel

This section proposes our specific relational database access control metamodel (RBAC-inspired). Next sections describe how to automatically populate it based on the actual policies enforced in a particular database and how it can be used to analyze those policies.

The SQL standard predefines a set of privileges and object types that can be used in the definition of a relational database. Our metamodel provides a direct representation of these concepts to facilitate the understanding of the security policies linking the security elements with the schema objects constrained by them as depicted in figure 6.1. In the following, we describe the main elements of the metamodel.

Database objects: The main objects users can be granted privileges on are the following: tables, columns, views, procedures and triggers. Each one of these elements is represented in the metamodel by a metaclass inheriting from the *SchemaObject* metaclass. We remark that views, as can be seen in metaclass *View*, can have not

only columns corresponding to table columns but also derived columns. Databases include a set of schemas (*Schema* metaclass) which in turn contain the *SchemaObjects*.

Privileges: Privileges, divided in five categories as presented in section ??, are depicted in the bottom part of figure 6.1. If needed, an extension of this metamodel (by inheriting from the *Operation* metaclass) could be provided to deal with vendor-specific privileges.

Subjects: Subjects executing actions on the DBMS are users and roles and they are, as such, represented in the metamodel with the corresponding metaclasses *User* and *Role*. The metamodel has to be also able to represent role hierarchies, normally allowed in DBMSs supporting roles. In the metamodel, the metaclass *Role* inherits from the metaclass *Subject* which enables it to become grantee. The *User* metaclass also inherits from *Subject* what means that, privileges, in contrast to pure RBAC, can be granted to both users and roles¹.

Other elements and constraints: There are several other aspects that have to be taken into account when developing a domain specific metamodel for access control in DBMSs.

The execution of privileges may need to be constrained. In DBMSs this is normally achieved by using triggers and procedural code. The metamodel should permit the representation of the existence of such constraints. The metaclass *Constraint* meets that purpose. Typically it points to a *Trigger* linked to the object and the operation on it that is constrained by the trigger. The *Trigger* metaclass also includes the attributes the trigger body and the condition and error message to be displayed when the trigger execution fails due to any exception.

Another aspect to be taken into account is object ownership as it is the basis for permission delegation. An association between objects and subjects records this ownership relationship.

Finally, global permission, i.e., permissions that are granted on all the corresponding elements (e.g., granting SELECT permission on ANY TABLE gives permission to select over all the tables in the schema) also exist in DBMSs and must be represented. For that purpose, in the metamodel, permissions can be granted on any object, including the metaclasses *Schema* and *Database*. A select permission granted on *Schema* or *Database* represents that the grantee can select all the tables and views contained in those objects.

1. The model could be simplified by only allowing roles to have privileges and then creating and assigning a role to each user in the DBMSs. However, we consider better to represent privilege assignment as it happens in database systems.

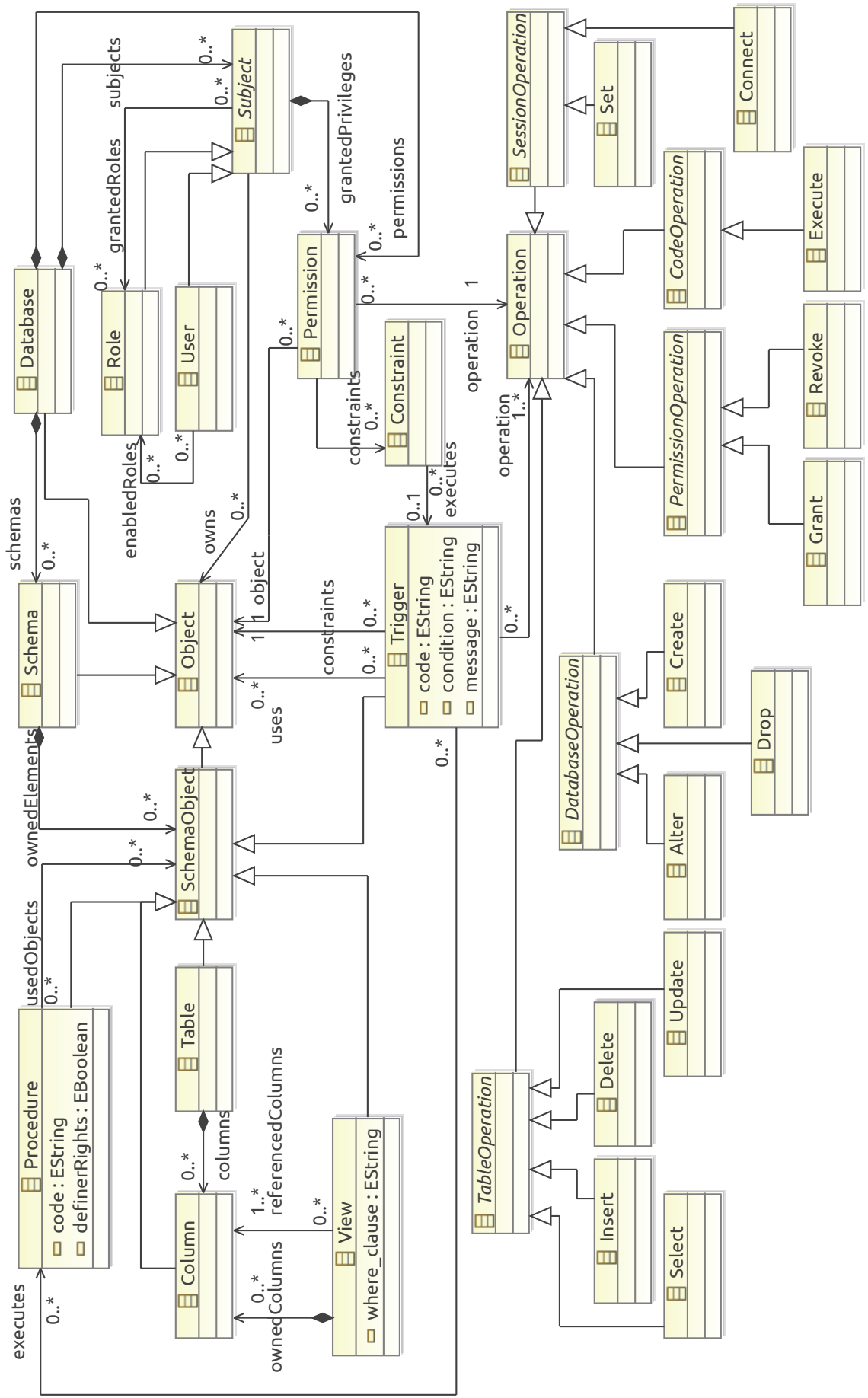


Figure 6.1: DBMS Security Metamodel

6.3 Reverse Engineering Process

The previous metamodel allows to represent database access control policies at the logical level. Models conforming to this metamodel express the security policies in place in a given database in a vendor-independent manner. These models can be manually created but ideally they should be automatically created as part of an automatic reverse engineering process that instantiates the model elements based on the information extracted out of the database. Note that, the extracted models are vendor-independent but the extraction process is not since each vendor uses different internal structures (i.e. a different set of tables/columns in the data dictionary to express this information). In fact, we could regard this “injection” process as the one that abstracts out the specific product details.

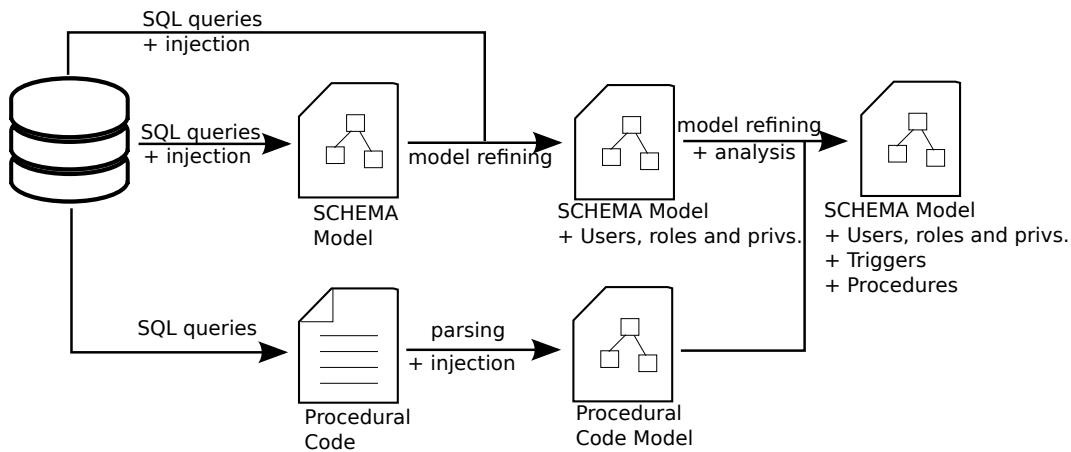


Figure 6.2: Extraction process

Our reverse-engineering approach is summarized in Figure 6.2. It starts by populating our security model with the basic schema information (tables, views, etc). Then this model is refined to add user, roles and privileges information whereas in a last step, the bodies of triggers and stored procedures are analyzed to complement the access control policies in the security model.

In the following we will detail the process for the reverse engineering of security policies over an Oracle 10g DBMS. This same process could be reused for other DBMSs changing the references to the specific data dictionary tables with the corresponding ones in that system.

6.3.1 Extracting general schema information

The first step of the reverse engineering process consists in populating the part of the model that describes the structure of the database itself e.g., schemas, tables (and columns), views, procedures, etc. In order to do this, an injector (in our terminology, an injector is a software component that bridges technical spaces[57],

in this case moving information from the database technical space to the modeling technical space. See section 2.2 in Chapter 2 for more details) is needed. This injector connects to the database and queries the dictionary tables to retrieve all the necessary information. The selected objects are inserted as model elements in the security model. In the concrete case of Oracle, our injector has to query the tables and views in listing 6.1.

View extraction is more challenging since we need to parse the view definition to get the list of tables (or other views), columns and conditions the view selects. Since the tables and columns have already been created in the model in the previous step, the result of the parsing is a set of links between the view object and the other objects filtered by the view. The *where* clause will be copied as it is into the *condition* attribute of the view metaclass. Moreover, a view can contain derived columns, e.g., columns that do not exist in any table like sums, avg, etc. To retrieve these columns, the database dictionary has to be queried to extract the list of columns of the view and then intersect it with the set of previously obtained columns (i.e., view columns pointing to table columns). Each column that is not in the intersection set will be created as a view column in the model.

Listing 6.1: database dictionary tables for general schema information

```
1 DBA_TABLES //List all the tables in the database including their definition text
2 DBA_VIEWS //List all the views in the database including their definition text
3 DBA_TAB_COLS //List all the columns of all the tables in the database
4 DBA_SCHEMAS //List all the schemas in the database
5 DBA_PROCEDURES //List all the stored procedures in the database
6 DBA_TRIGGERS //List all the triggers in the database
7 DBA_SOURCE // Source code of procedures and triggers
8
9 //Access control information
10 DBA_USERS //List all the users in the database
11 DBA_ROLES //List all the roles in the database
12 DBA_COL_PRIVS //Describes all column object grants in the database
13 DBA_ROLE_PRIVS //Describe all role grants to user or roles
14 DBA_TAB_PRIVS //Describe all object (tables, views, etc) grants in the database
```

6.3.2 Extracting users, roles and privileges

Once the schema information has been already inserted in the database security model, it is time to add the access control information. In listing 6.1 we show the tables that have to be queried to obtain all the information regarding users, roles, assignment of roles and assignment of privileges. As in the previous step, an injector is needed to query the database and populate the model with the results. After this step, the general access control information of the database, i.e., subjects, objects and permissions are already represented in the security model.

6.3.3 Extracting stored procedures and triggers information

Triggers. Complex security checks can be implemented by means of triggers that fire to prevent certain actions when performed in a certain context. However, triggers are used for a huge variety of tasks beyond security purposes. In our approach, all triggers are retrieved and parsed, then, they are analyzed with respect to a number of heuristic conditions in order to select which of them are implementing security checks and constraints and discard the rest.

The heuristic conditions analyze the following aspects:

- *Trigger kind:* Triggers are associated with statements (e.g., a select statement) and fired when the statements are invoked. However, it is possible to select if the trigger will be executed before or after the statement. *BEFORE STATEMENT* triggers are executed before the statement is completed, enabling the possibility of evaluating security concerns (e.g., the possibility to make inserts in certain tables could be enabled only to working days). Conversely, *AFTER STATEMENT* triggers are executed once the action of the statement is performed, so, when involved in security, they are normally used for logging purposes. Clearly, our focus should be in the *BEFORE STATEMENT* triggers.
- *Trigger contents:* Although the kind of the trigger is an important hint, it is not enough to decide if it implements a security check or constraint or any other operation (For example, for each table with an auto-generated primary key, this will be implemented as a *BEFORE STATEMENT* insert trigger. Similarly, a number of derived attributes or inter-table check constraints may be implemented with this kind of trigger). To tackle this problem, we analyze the contents of the trigger in order to find operations that are likely to be used when performing security checks like system context information checks, user information checks, exception raising's, etc.

As a summary, a trigger will qualify as a *security trigger* if it fulfills all the following heuristic conditions:

1. The trigger is a before statement trigger.
2. The trigger contains an exception section that raises an exception.
3. The trigger evaluates conditions on the system (ip address, host, time) or user information (name, assigned privileges). Note that checking this part is strongly vendor dependant.

As an example, listing 6.2 shows two triggers from the well-know Human Resources (HR)² schema example provided by Oracle: *Secure_employees* and *Update_job_history*. The latter one is an *AFTER STATEMENT* trigger which directly disqualifies it as a security enforcement trigger. Conversely, the former fulfils all the heuristic conditions. It is a *BEFORE STATEMENT* trigger, it raises an exception and checks system information (the time) and thus it qualifies as a *security trigger*.

Listing 6.2: Trigger's code

```

1 TRIGGER update_job_history
2   AFTER UPDATE OF job_id, department_id ON employees
3
4   FOR EACH ROW
5   BEGIN
6       add_job_history(:old.employee_id, :old.hire_date, sysdate,
7           :old.job_id, :old.department_id);
8   END;
9
10 TRIGGER secure_employees
11  BEFORE INSERT OR UPDATE OR DELETE ON employees
12
13  BEGIN
14      IF TO_CHAR (SYSDATE, 'HH24:MI') NOT BETWEEN '08:00' AND '18:00'
15         OR TO_CHAR (SYSDATE, 'DY') IN ('SAT', 'SUN') THEN
16          RAISE_APPLICATION_ERROR (-20205,
17              'You may only make changes during normal office hours');
18      END IF;
19  END secure_employees;

```

Once a trigger is identified as a *security trigger*, the constraints imposed by the trigger need to be extracted and added to the model (see section 6.5).

Stored procedures. As introduced in section 6.2, stored procedures can be executed with invoker or definer's rights. In the latter, the invoker role gets transitive access to certain database objects. The source code of the procedure must be analysed to obtain such set of transitive permissions. Our approach parses the store procedures and extracts the accessed database objects. These are then linked to the procedure in the database security model in order to easily retrieve them later on during the analysis phase. As an example, the *add_job_history* procedure in listing 6.3 is declared to be invoked with definer rights. The table it accesses, *Job_history*, would appear linked to this procedure in the security model.

6.4 Applications

The database security model obtained at the end of the previous step can be used in many different scenarios. In the following we describe some relevant ones. Note that the implementation of these scenarios benefits from the fact that at this stage

2. http://docs.oracle.com/cd/B13789_01/server.101/b10771/scripts003.htm

Listing 6.3: Procedure code

```

21
22 PROCEDURE add_job_history(
23     p_emp_id job_history.employee_id%type
24     , p_start_date job_history.start_date%type
25     , p_end_date job_history.end_date%type
26     , p_job_id job_history.job_id%type
27     , p_department_id job_history.department_id%type
28     )
29
30 IS
31 BEGIN
32     INSERT INTO job_history
33         (employee_id, start_date, end_date,
34          job_id, department_id)
35     VALUES(p_emp_id, p_start_date,
36            p_end_date, p_job_id,
37            p_department_id);
38 END add_job_history;

```

security aspects are expressed at the model level and thus all existing model-driven techniques can be reused when manipulating them. This model-based representation also allows us to get rid of all vendor specific knowledge during the abstraction process. Therefore, these operations can be applied no matter the database system in place.

In order to illustrate the model analysis and manipulation that follows in this section, we will use the Human Resources (HR) schema.

The HR schema contains:

- 7 tables: *Countries*, *Departments*, *Employees*, *Jobs*, *Job_history*, *Locations*, *Regions* that build up the *Human Resources* database.
- 1 view: *Emp_details_view*, that summarizes the employee information contained in all the other tables.
- 2 stored procedures: *Add_job_history* to insert values into the *Job_history* table, and *Secure_dml* that assures that changes can not be performed out of the office working hours.
- 2 triggers: *Secure_employees* that calls *Secure_dml* to prevent unauthorized modifications in the *employees* table and *Update_job_history* that keeps *Job_history* up to date with respect to relevant changes in the *employee* table.

To that example we have added two roles, *HR_MANAGER* and *HR_TRAINEE* so that we can show the capabilities of our metamodel and reverse-engineering process to obtain and represent database RBAC security policies. Then, we have applied our tool to obtain the corresponding database security model, hereinafter called HR security model.

Visualization. Visual data is often easier and faster to analyze than textual or tabular data. Using MDE tools we can easily provide a visualization of our database security model so that the relation between subjects, objects and permissions can be easily grasp. We have use Portolan³, a model visualization tool, in order to generate such visualization.

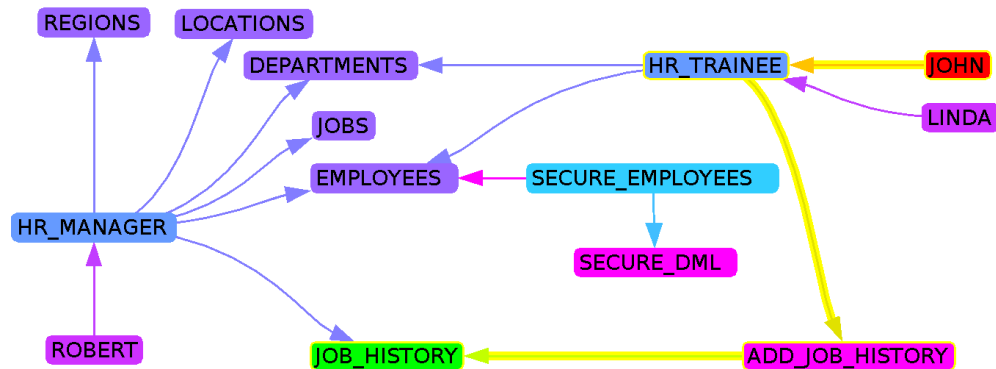


Figure 6.3: Database security model visualization

In Figure 6.3 we show the visual representation we obtained for the HR security model, where each kind of element and relation is shown in a different color. Thanks to this visualization we can quickly see that HR_MANAGER has privileges in all tables whereas HR_TRAINEE has only privileges on the *EMPLOYEES* and *DEPARTMENTS* tables and on the *ADD_JOB_HISTORY* stored procedure. Moreover, the table *EMPLOYEES* has a constraint implemented by the trigger *SECURE_EMPLOYEES* (that calls an stored procedure). Finally, we used the portolan path-discovery feature to see if there is a path between the database user *JOHN* and the table *JOB_HISTORY*. This path exists, as the table is reachable through the stored procedure *ADD_JOB_HISTORY* (note that for simplicity and readability, some information like ownership or system roles is not shown in the diagram).

Complex security queries and metrics. The most basic thing we may want to do with a security model is to query it to learn more about specific details of the security policies currently enforced in the database.

This is very complex to do directly on the database itself since this information is scattered among a number of database dictionary tables which are completely vendor-specific. Instead, when using our model we can just use a standard model query language to traverse the information in the extracted model classes. The model query is defined just once and can be executed on security models extracted from any relational vendor.

As an example, we have used the Eclipse⁴ environment to query the HR security

3. <http://code.google.com/a/eclipseorg/p/portolan/>

4. <http://www.eclipse.org>

Listing 6.4: OCL Query Example

Evaluating:

```
let constrainedPrivileges : Sequence(DBSecurity::Permission) =
  self.subjects->select(e | e.name = 'HR_MANAGER')
  ->collect(e | e.grantedPrivileges)
  ->select(e | not e.constraints->isEmpty()) in
  constrainedPrivileges
  ->collect(e | Tuple{object = e.object,
                    operation = e.operation,
                    type = e.constraints.executes->first() })
```

Results:

```
Tuple{object=Table EMPLOYEES, operation=Update, type=Trigger SECURE_EMPLOYEES}
Tuple{object=Table EMPLOYEES, operation=Insert, type=Trigger SECURE_EMPLOYEES}
Tuple{object=Table EMPLOYEES, operation=Delete, type=Trigger SECURE_EMPLOYEES}
```

model. In listing 8.5 we show a query written in OCL[71], the standard query language for models. It extracts, for a given role (in this case HR_MANAGER), all the granted privileges that are subject to constraints (e.g. there is a trigger that restricts the execution of that privilege). As a result the query show the privilege, the object and the trigger enforcing the constraint.

In the same way, we can also calculate metrics to learn more about the implemented security policy. A software metric is a measure of some property of a piece of software or its specifications. Quantitative measurements according to these metrics could help in several activities, like quality assurance and feature analysis. Clearly, there are many metrics that may be interesting to evaluate the security policies of an organization. Measures like counting the average of objects reachable per subject or how many users are assigned per role can give as one idea of how the policy is defined. We can discover, by example, that some roles accumulate too many privileges what could put the system into risk.

In the MDE world, the Object Management Group (OMG) has proposed a standard metamodel, Software Metrics Metamodel (SMM) to express the metrics and measurements on a model. SMM is integrated in the model-driven reverse engineering framework MoDisco[26]. Therefore, we can easily define SMM metrics and evaluate them on a given security model to quickly get some summary information. For instance, figure 6.4 shows the (tree-based representation) SMM model for the metric “calculate the average number of objects accessible per role”. The metric first counts the select privileges that are granted to roles, then groups them by role and finally calculates their average number.

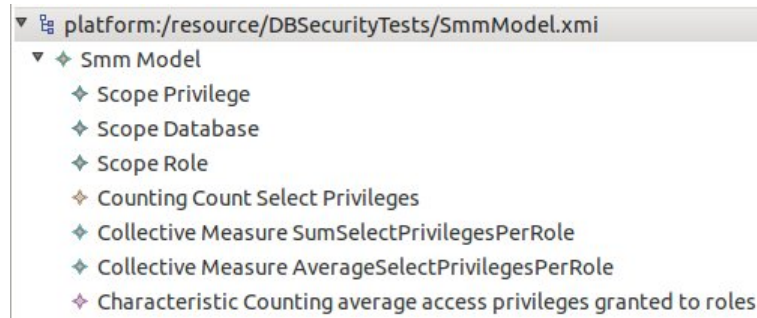


Figure 6.4: SMM Metric Example

In scenarios where an objective policy is provided, using model set operations can also provide useful information. For instance, model difference [11, 54] is an operation between two models, M_1 and M_2 that calculates the differences between them, giving as a result a model (or any other representation) containing either elements that are different or elements that only exist in one of the models. The union of models is the opposite operation and supposes the merge of the elements of two models. Similar definitions can be obtained for all the typical set operations.

Thanks to these operations we could for instance see if a database D_a has more strict policies than a database D_b (e.g. $M_{D_b} - M_{D_a}$ would reveal those permissions in D_b not present in D_a) or create the security policy for a new database D_c resulting from the merge of two databases D_a and D_b either by stating that the security policies in the new database are the union of those in D_a and D_b (D_c would be less restricted) or their intersection (D_c would become more restricted).

Refactoring of security policies. The complexity and frequent evolution of access control security policies may lead to correct but non-optimal definitions. This problem, can be tackled by using a technique, refactoring, proved useful for quality improvements in several domains (source code, models, etc).

A refactoring on the database security model will keep the semantics of the policy, i.e., the same objects will be accessible by the same subjects, while improving one or more quality attributes of the model. Attributes like brevity, clarity and breadth (how comprehensive a policy is) of security policies [45] can significantly impact the quality of the policy as they will ease the task of managing it.

With our approach we could build a library of security refactoring patterns for databases that would allow designers to improve the security implementation of any database. As an example, we could envision an `IntroduceRole` refactoring that creates a new role grouping a set of users with the same permissions. This refactoring does not change the behaviour of the policy (i.e. users do not have more nor less permissions than before) but will improve its maintainability since now we can work at the role-level instead of managing every time many individual users.

Refactorings can be implemented with model transformations whereas refactor-

Listing 6.5: refactoring example

```

1
2 module refactoring;
3
4 create OUT : SecurityDB refining IN : SecurityDB;
5
6 rule IntroduceRole{
7   from
8     user: SecurityDB!User
9   to
10    modifiedUser: SecurityDB!User (
11      permissions <- Sequence {} ,
12      roles <- user.grantedRoles->append(newRole) ),
13    newRole: SecurityDB!Role (
14      name <- 'newRole' ,
15      permissions <- user.permissions )
16 }

```

ings opportunities (“smells”) can be specified by means of model queries. In listing 6.5 we show a Model-to-Model transformation excerpt implemented in the ATL transformation language that introduces a new role, assigns to it the permissions of a given user, remove the permissions from that user and grant her the new role.

Security policy deployment (reengineering). New requirements to be met by the database security policy may appear. Instead of directly modifying the database security implementation, what will require vendor-specific knowledge and could lead to introduce errors as stated in the motivation of this work, the extracted model can be used to start a model-driven forward engineering process.

First, our reverse engineering process would create the security model out of the current database implementation. Then, the required changes and validations will be performed on the security model. Finally, model-to-model transformation and model-to-text transformations will be used to generate the SQL sentences and procedural code corresponding to the evolved model.

Our framework can also be used as part of existing forward engineering methods [70, 17, 91] able to derive policy implementations from high-level security specifications. All these approaches could use our metamodel as a pivot/intermediate representation in their “code-generation” process to simplify bridging the gap between the high-level specifications and the vendor specific notation.

6.5 Tool Support

In order to validate the feasibility of our approach, a prototype tool has been developed and can be found online.⁵

The tool has been implemented on top of the Eclipse Modeling Framework

5. http://www.emn.fr/z-info/atlanmod/index.php/DBMS_security_reverse_engineering

(EMF) [92] the de-facto standard for implementing model-driven techniques in the Eclipse platform.

More specifically, the proposed security database metamodel has been implemented as a EMF (Ecore) metamodel. For each metamodel, EMF automatically generates a tree-based model editor and a Java API to manipulate models conforming to that metamodel. A reflective API to manage generic models is also available.

Then, the injectors to generate the security model from the database dictionary have been implemented in Java as a set of queries to the Oracle 10g database dictionary. The EMF reflective API is used to process the results of the queries and create the corresponding model elements and links in the security model.

Finally, XText⁶, an eclipse-based framework for building domain specific languages, has been used to parse and inject into models the source code of triggers and stored procedures based on our grammar for the Oracle PL/SQL language (defined as an extension of a publicly available one⁷). To implement the trigger discrimination, condition extraction and condition representation we have used ATL model-to-model transformations.

6. Xtext. <http://www.eclipse.org/Xtext/>

7. <http://svn.code.sf.net/p/plsql-xtext/code/>

Extracting AC Models from Content Management Systems

Web Content Management Systems (WCMSs) is the technology of choice for the development of millions¹ of Internet sites and increasingly, becoming a framework widely used for the development of Web applications. They provide an integrated environment for the definition of the design, layout, organization and content management of the application and, because of its relative ease of use, they enable users with little technical knowledge to develop fully functional systems.

This widespread use highlights the importance of security requirements, as WCMSs may manage sensitive information whose disclosure could lead to monetary and reputation losses. Due to the nature of the users, the focus has often been in facilitating the WCMSs configuration. Although these systems are easy to use, a proper configuration is needed to minimize the introduction of vulnerabilities. As a consequence, tools for checking the configuration of WCMSs have been provided and analysed by the scientific communities. However, these tools are focused in low-level security aspects like management of cookies or prevention of SQL injection vulnerabilities [66, 97].

Moreover, despite some approaches for extracting AC information from dynamic web applications source code[44, 10], little attention has been brought to the analysis of how developers use the content protections mechanisms provided by WCMSs systems. Particularly, Access-control techniques, integrated in most WCMSs and capable of enforcing confidentiality and integrity of data must be analyzed so that no logical flaws are present in the security policy. Unfortunately

1. <http://trends.builtwith.com/cms> (15 May 2014)

knowing if an implemented AC policy on a WCMS provides the required content protection is a complex and error-prone task as the specificities of each WCMS vendor AC implementation must be mastered (e.g. the set of roles and permissions that can be defined vary largely among the different WCMSs).

In order to tackle this problem, we propose to raise the level of abstraction of the AC implementation so that it gets represented according to a vendor-independent metamodel. This WCMS security metamodel must be able to represent WCMS specific information along with AC concerns. We can regard such a metamodel as an extension of typical AC models specially tailored to the representation of security in WCMSs.

Ideally, these models should be automatically obtained from existing WCMS AC configurations. Therefore, here, along with the description of the metamodel for the representation of WCMS AC policies, we describe the process to automatically extract them from a Drupal[6] WCMS, one of the three most popular WCMSs. Note however, that the extraction from other WCMSs like Wordpress or Joomla will follow the same process. Once these models are available, they can be analysed in a generic way, focusing in the security aspects and disregarding the specificities of concrete vendors. Moreover, Model-driven tools for querying, performing metrics, provide visualizations, etc, become automatically available, easing the analysis tasks.

Combining the vendor-independent representation and the extraction process, migration and reengineering tasks are facilitated. Recovered AC policies represented in our metamodel can be used, after its analysis, correction, etc., as a pivot representation for automatically generating correct configurations or configurations for other WCMSs.

7.1 Motivation

WCMSs are Content Management Systems (CMSs) specially tailored to the authoring of content in the Internet. They integrate facilities for the definition of the design, layout, organization and collaborative content management of web sites and can also be used, due to the wide range of features they offer, as a framework basis for the development of web applications. They are, due to its relative ease of use (they allow users with little knowledge of web markup and programming languages to create and manage fully functional web sites) and low cost, the technology of choice for the development of millions of web sites.

In general, they are composed by a back-end, comprising the repository of contents and administrative tools and a front-end that displays this information to web clients.

As discussed in the introduction, security is a critical concern in WCMSs as they may manage sensitive information. Therefore, security mechanisms have been integrated in most WCMSs where access-control mechanisms play a prominent role. However, WCMSs users often lack depth technical and security knowledge, so that the implemented access-control policies may contain security flaws. For instance, in Drupal, the permission *Delete any content* of type Article could be, by mistake, granted to the default role *Authorized User*. As all user-defined roles inherit by default from this role, this mistake will give them the capacity of deleting the content of the application. Furthermore, the frequent need of migrating from one WCMS to another, also highlights the need for understanding the current security policy so that it can be accurately translated especially, since the security concepts in each WCMS differ. Failing at doing so will often imply putting the new system under risk.

In this scenario, analysing and understanding the security policy enforced by a WCMSs turns up as a critical necessity. Unfortunately, each WCMSs vendor provides its own access-control model and management tools so that these analysis tasks requires in-depth knowledge of the concrete system in hand.

We believe that, in order to tackle this problem, the level of abstraction of the AC policies enforced in WCMSs needs to be raised, so that the information is represented in a vendor-independent manner. We propose thus to represent the AC policies as models corresponding to a WCMS access-control metamodel. In the rest of the paper, we will detail the proposed metamodel and extraction approach.

7.2 WCMS Access-Control Metamodel

Central to the process of recovering and analysing the access-control information of WCMSs is the definition of a metamodel able to concisely represent the extracted Access-control information in the domain of WCMSs. This metamodel must also be platform-independent, so that we can analyse the access-control information disregarding the especificities of the concrete WCMS security features and implementation.

Figure 7.1 depicts our proposal for such a metamodel. It is an RBAC-inspired metamodel, thus, containing all RBAC basic concepts along with WCMS specific information. It consists basically of four kind of elements. Contents, i.e., the information hosted by the system, Actions, i.e., operations that can be performed on the WCMS, Permissions, i.e., the right of performing these Actions and Subjects, i.e., the triggers of Actions. In the following, we will detail the metamodel concepts of these categories along with its rationale.

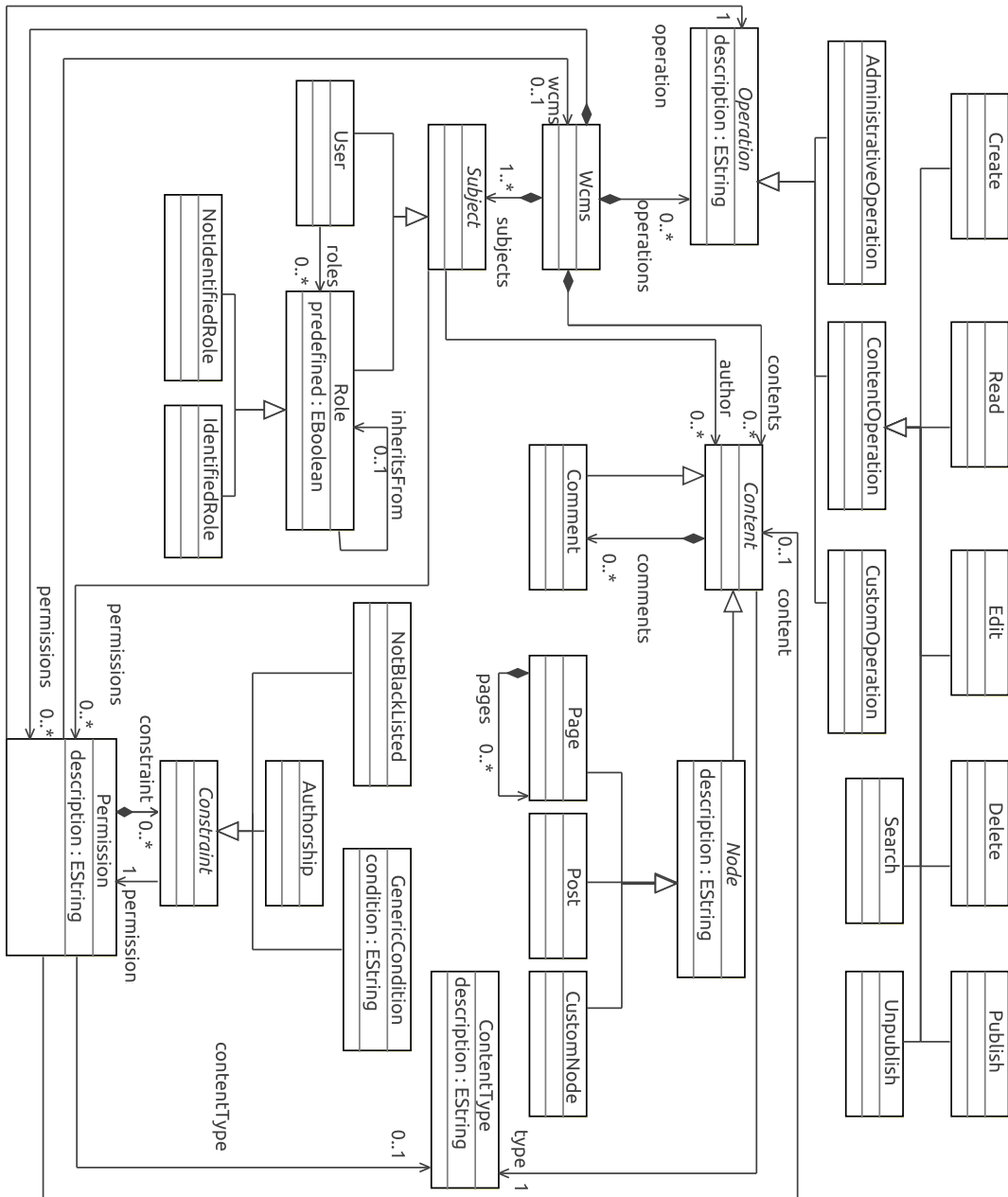


Figure 7.1: WCMs metamodel excerpt

7.2.1 Content

The content of a WCMS is the information it manages. This is represented in our metamodel by the *Content* metaclass. Each WCMS defines its own kinds of content. To be able to represent that eventuality, our *Content* elements have a *ContentType* that identifies its type. This also allows for the representation of fine-grained and coarse grained access-control. Effectively, some WCMS access-control models allow for the definition of different permissions for individual content elements, while others only allow the definition of global permissions on all the contents of a type.

Then, we provide the users of our metamodel with some predefined kinds of content. In one side we have *Node*, representing the principal contents of the WCMSs. We have specialised them in two subclasses: *Page* that represents full content pages (that can contain other pages) and *Post* that represents individual blog posts. We also provide a *CustomNode* metaclass so that additional types of nodes can be integrated. On the other side, we have *Comments* that represents comments that can be posted in any other content element. We do not represent pages in the back-end of the WCMS used to administer content. That behaviour will be represented by permissions of executing administrative operations on the WCMS.

7.2.2 Operations

Operations are the actions that can be performed over the *WCMS*. We can divide all operations that can be done over a WCMS between two types, content operations and administration operations (e.g., operations to manage users and roles). The latter category is more WCMS specific and as such, we will uniquely represent the permissions on that category with the metaclass *Administration Operation*.

W.r.t. the content operations, in our metamodel, all CRUD actions are available: *Create* for the creation *Content* elements; *Edit* for the modification of already created *Content* elements; *Read* for reading/viewing created *Content* elements; *Delete* for the deletion of *Content* elements. The *Search* operation is also available. It is a very common action in WCMSs and, as it can be expensive, it is usually restricted only to certain users (e.g., logged users). Additionally, there are two special actions the *Subjects* of WCMSs can perform. *Publish* and *Unpublish*. In WCMSs it will not be surprising to find that some *Subjects* can create and manage contents using CRUD operations while not being authorized to make it publicly available without revision-moderation from another authorized *Subject*. These two actions support this behaviour. *Publish* is the action of making available some created *Content* ele-

ment and *Unpublish* is the action of removing a piece of *Content* from its place of publication without internally deleting it.

Finally, we also consider the possibility of new operations that may appear e.g., when extending the WCMS. In order to be able to represent these possible new operations, we provide the *Custom Operation* metaclass. This way, if the WCMS is extended with the capability of e.g., doing polls, an eventual new operation, vote, could be represented by this metaclass.

7.2.3 Permissions

Permissions are the right of performing actions on the WCMS. They can define *constraints* that restrict the *Permission* to execute the corresponding action only when certain conditions hold. In our metamodel, we have identified two kinds of *Constraints* that typically appear in WCMSs: *Authorship* and *NotBlacklisting*. The former expresses that the permission is effective only if the *Subject* is the author of the *Content* whereas the latter restricts the applicability of the permission to the condition of not being blacklisted. Other conditions may exist and therefore, we provide the means to represent them by the *GenericCondition* metaclass. It holds in a text field the condition of the *Constraint*. The nature of the contents of this text field is left open to the metamodel users, so that it can hold conditions expressed in natural language or in more formal constraint languages like OCL. Similarly, in [76] the authors added the constraints to permissions represented by triggers to a metamodel tailored to represent Relation Database Management Systems (RDBMS) access-control by adding the source code of the triggers. As in there, the representation and extraction of the meaning of such custom constraints will require a further analysis. We leave such analysis as future work.

7.2.4 Subjects

are the elements interacting with the contents of the WCMS by performing actions (note that a *Subject* can be the author of a piece of *Content* and that this may influence the Access-control of the information. Thus, this is represented in our metamodel). Following a RBAC approach, in our metamodel we have two kinds of Subjects: *Users* and *Roles* where *Users* get *Roles* assigned. However, unlike RBAC we are more flexible in the permission assignment by allowing both *User* and *Role* to get permissions granted.

Depending of the WCMS at hand, *Roles* are predefined by the application or can be defined by the developer. Both cases can be discerned in our metamodel by using

the *predefined* attribute of the *Role* metaclass. Moreover, we have identified two specific roles that often appear in WCMSs. *IdentifiedRole* and *NotIdentifiedRole* are often present in WCMS to discriminate between not logged and logged users. As such, we have decided to add them to the metamodel so that this behaviour can be easily modeled. Finally, role inheritance is also supported.

7.3 Approach

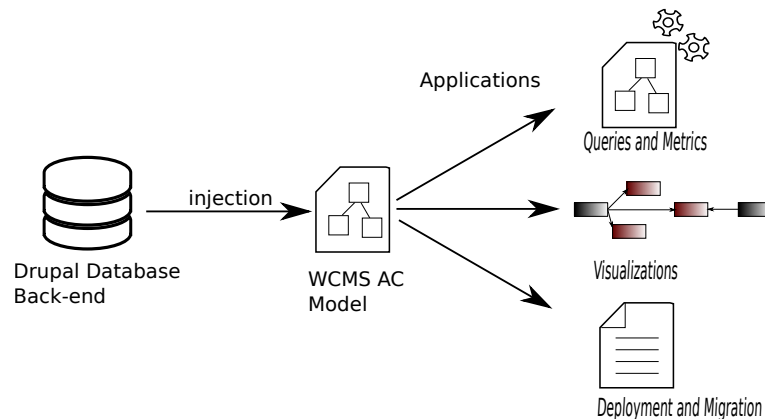


Figure 7.2: Drupal AC extraction and analysis approach

Although our metamodel could be manually filled by inspecting the AC information using the WCMS administration tools, ideally, it should be filled by an automatic reverse-engineering approach. In the following, we present such an automatic process for a Drupal WCMS although it can be easily adapted to work with other WCMSs. The process is depicted in Figure 7.2.

In Drupal, contents, along with the corresponding access-control information (i.e., users, roles and permissions) are stored in a database back-end. Thus, in order to obtain a model conforming to our WCMSs metamodel with this information, an injection process needs to be launched. This process performs SQL queries over the database back-end while creating, as output, the corresponding model elements. Note that, additionally, extra access-control rules could be defined or modified programmatically, in the source code of plugins, etc. Techniques as the ones in [44, 10] could be used to as a further step to complement our approach.

Note however, that this first step will require a previous step, i.e., the discovery of the data model of the WCMSs as each WCMS defines its own. For doing so, we can rely on the WCMS available documentation or in worse cases, to schema extraction tools. In the case of Drupal, the relevant tables are the following ones: `USERS`, that contains all system users; `ROLE`, containing all roles; `USER_ROLES` relating users with assigned roles; `PERMISSION` connecting roles with permissions; `COMMENT` for the special content of the type comment and `NODE` for all

the other content types.

Drupal AC. Extraction evaluation: In drupal, there exist three main kinds of content, Pages, Articles and Comments and three roles by default, *Anonimous*, *Authenticated* and *Administrator*. By default, any new user-created role, what is allowed, inherits the permissions from the *Authenticated* one. Thus, to create roles more restricted than the *Authenticated* role, the *Authenticated* role needs to get permissions removed. Using the default modules, permissions can not be granted in concrete content, e.g., concrete pages but on content types. This way, permission to edit content can be granted and all Pages but not on individual pages (apart from the distinctions made wrt to ownership and publish/unpublished).

Our metamodel is capable of representing the AC information of Drupal by using an injector performing the mappings summarized in Table 7.1.

DRUPAL	WCMS Metamodel
User	User
Default Role	The Anonymous role to NonIdentified role, Administrator and Authenticated role to IdentifiedRole with inheritance relation between the translation of Administrator and Authenticated roles
User-defined Role	IdentifiedRole with inheritance relation to the translation of the default role Authenticated Role
Page and Article types	ContentType for Page and Article
Node	If the type is page or article, Page with the proper ContentType. If it is blog post, Post. If there is another type of node, CustomNode
Content actions	ContentOperation.
Administrative actions	AdministrativeOperation.
Comment	Comment pointing to the corresponding Content
Permission	Permission with the corresponding links to the subject, object and operation. For content permissions (or administrative permissions on content), link to Content or ContentType (for permissions granted on all content of a given type). For administrative permissions (except for the administrative permissions on content), link to the WCMS instance.

Table 7.1: Mapping from drupal to our metamodel

7.4 Applications

Once the injection process is finished, we can start analysing and manipulating the obtained model in a vendor-independent way.

In order to ease the discussion, we present here a small example on a Drupal installation intended to work as a blog. Notably, and in order to show , 1) the access-control capabilities of the the CMS and 2) the ability of our metamodel to capture the AC implementation on Drupal, custom users and roles have been created.

Concretely, two custom roles have been created. The Editor role, intended to create contents on the CMS and the Reviewer role, intended only for those authorized roles with the rights of publishing comments. Note that in Drupal, as discussed in the previous section, all custom roles inherit from the Authorized role. Thus, in order to be able to give the least privileges possible to our custom role, the authorized role have been configure to hold only the permissions to read content and to write comments. This way, the Reviewer role will inherit all its privileges through this inheritance, while the Editor role is explicitly granted the createPost permission. Then, two users, SiteEditor and SiteReviewer are created and assigned to the Editor and Reviewer roles respectively.

In the rest of this section, we will summarize some applications of our WCMS access-control model.

Visualization: Visual data is often easier and faster to analyze than textual or tabular data. Using MDE tools we can easily provide a visualization of our WCMS AC model so that the relation between subjects, objects and permissions can be easily grasp. As we have done in Chapters 5 and 7 for the domains of firewall networks and relational databases respectively, we have used Portolan in order to obtain visualizations of our model. Concretely a transformation between our security model and the Portolan Cartography model have been performed in order to show the information regarding Users, Roles and Privileges.

In Figure 7.3 we can see the output we have obtained on a model extracted from our proposed Drupal example (note however that the model have been pruned to facilitate the discussion, so that only the relevant permissions for the example are depicted).

By inspecting the obtained graph, we can easily distinguish which roles are the default ones and which ones are custom created, their inheritance relationships, the user role assignment and the privilege assignment.

Queries: The most basic thing we may want to do with a security model is to query it to learn more about specific details of the security policies currently enforced in the WCMS. As an example, we could want to know what elements can be accessed

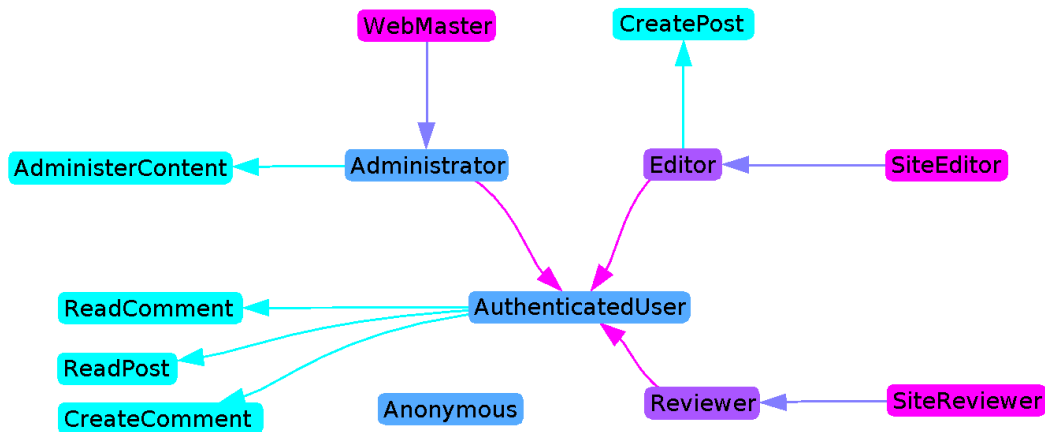


Figure 7.3: WCMS security model visualization

Listing 7.1: OCL Query Example

Evaluating:

```
let Editor : wcms::User = self.subjects->select(e | e.ocIsTypeOf(wcms::User))
->select(e | e.name = 'SiteEditor')->first().oclAsType(wcms::User) in

Editor.permissions->asSequence()
->union(Editor.roles->closure(e | e.inheritsFrom)->collect(e | e.permissions)->
  asSequence())
->union(Editor.roles->collect(e | e.permissions))
```

Results:

```
Permission CreateComment
Permission ReadPost
Permission ReadComment
Permission CreatePost
```

by a given user, taking into account its assigned roles and also the permissions inherited from parent roles. This is very complex to do directly on the WCMS itself since this information is scattered among a number of database tables which are completely vendor-specific. Instead, when using our model we can just use a standard model query language to traverse the information in the extracted model classes. The model query is defined just once and can be executed on security models extracted from any relational vendor.

In Listing 7.1 we show such an OCL query. Given a concrete user (the SiteEditor user), we obtain all the assigned privileges. We do this by concatenating the privileges directly granted to the user (what returns and empty list in the case of Drupal), the privileges granted to the directly assigned roles (for our example, the CreatePost permission) and the privileges obtained by the inheritance relations of its assigned roles (for our example, the permissions obtained from the Authorized role, e.g., the CreateComment, ReadPost and ReadComment permissions).

WCMS migration: New requirements to be met by the application, discovered security vulnerabilities, technological choices, etc., may impose the migration from one WCMS to another. In this scenario, properly migrating the access-control information (users, roles, permissions) becomes critical. Our metamodel can be used as a pivot representation. Representing the AC information of the old WCMS in a model corresponding with our metamodel will facilitate its understanding and analysis, thus, helping to provide a good translation towards the AC model in the new WCMS.

Integration

8.1 AC Integration for multi-layer systems

As introduced in Chapter 2, Nowadays systems are often composed of a number of interacting heterogeneous subsystems. Access-control is pervasive with respect to this architecture, so that we can find access-control enforcement in different components placed at different architectural levels. Moreover, the access-control techniques implemented in each different component can follow different models (AC lists, RBAC, DAC, etc.) in order to best suit the needs of the component. Therefore, in any system, a set of different access control policies are enforcing the security goals. But these policies are not independent and relations exist between them, as relations exist between components situated in different architecture layers. Concretely, dependency relations exist between access-control policies, so that the effects of rules in a policy will depend on the decisions of rules in another policy.

Thus, ideally, a global policy representing the access-control of the whole system should be available, as analysing a policy in isolation does not provide enough information. However, normally, this global policy only exist in an implicit and not always consistent manner. Consequently, integration mechanisms are needed in order to 1) facilitate the analysis of the security status of the whole system and 2) achieve the global security goal.

The main focus of this chapter is therefore to provide a framework to integrate policies from different concrete components collaborating in an information system in a single policy. Two requirements need to be met for achieving the integration goal: The use of a common access-control policy language for representing the policies of each component and the recovery/representation of the implicit dependency

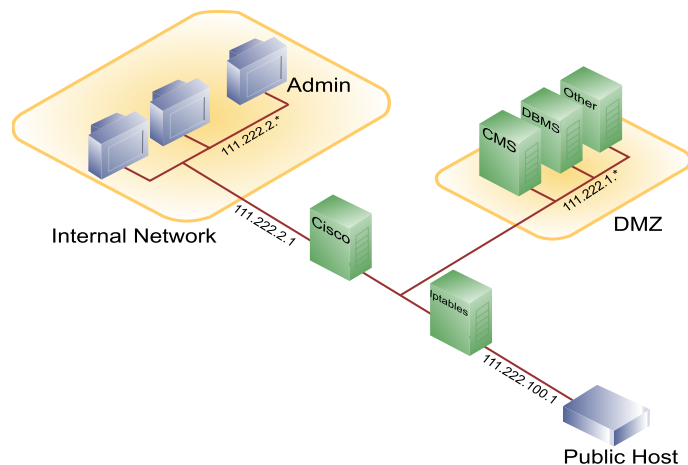


Figure 8.1: Information System Architecture

relations between them.

Translating all the recovered access-control policies to the same policy language, thus, representing them in a uniform way, eases the manipulation and reusability of analysis operations. However, domain-specific information may be present in the original policy representation, information that must be kept in order to keep the expressibility level. To meet both requirements, the flexibility to represent different policies and the ability to incorporate domain-specific concepts, XACML[59], an OASIS standardized language has been chosen.

In our approach, the extracted component policies will be translated to the XACML policy language while domain-specific information is added/kept by the use of profiles.

While the previous translation process provides advantages on its own, the policies remain isolated. Thus, we complete the integration framework with a semi-automatic process for detecting the policy dependencies and to organize the policies within a single XACML model. This enables us to see the policies in our information systems as a whole. We also provide a set of OCL[71] operations making use of it, so that complex analysis tasks, like inter architecture level anomaly detection, are eased.

8.2 Motivation

In order to motivate our approach, we reproduce here the Information System (IS) example introduced in Chapter 2, Section, 2.1.1. It will be used throughout the rest of the chapter.

As mentioned, the IS depicted in Figure 8.1 is a simple, yet very common information system. This IS is composed of several components working in different architecture layers, namely, a network layer, providing networking services and

enforcing access control through the firewalls (using Rule-based lists implementing Non-discretionary AC), a database layer, providing storage services and implementing role-based access-control (RBAC) through its built-in permissions schema and an application level, where a Content Management System (CMS) provides publication services. This CMS also enforces role-based access-control by using a built-in permission schema.

As we can see, three different systems enforce access-control. These systems are not isolated but collaborate to build up the functionality of a global system that encompasses them. Concretely, and in the case of subsystems located in different architecture layers, the collaboration relation is a dependency relation where systems in higher layers depend on services provided by lower layers. Access-control reproduces this behaviour.

Considering access-control rules as functions where a decision is taken w.r.t. to a subject accessing a resource to perform a given action under certain conditions and having the following form:

$$R(\textit{Subject}, \textit{Resource}, \textit{Action}, [\textit{Condition}]) \rightarrow \textit{Decision}$$

Let us take a look to the following examples:

Example 1:

$$R_{DB}(\textit{RoleX}, \textit{TableX}, \textit{Write}, 8:00 - 16:00) \rightarrow \textit{accept}$$

$$R_{FW}(\textit{Local}, \textit{DBServer}, \textit{Send/receive}, 8:00 - 14:00) \rightarrow \textit{accept}$$

In this example, a given role is granted permission to access a table for modification between 8:00 and 16:00. However, the access to the database server is constrained by a firewall rule, that only allows local access to the server between 8:00 and 14:00. As the database policy depends on the firewall policy, when the latter is more restrictive, it prevails. When asking if the role can access the table under which conditions, both policies need to be taken into account in order to provide a complete answer.

Example 2:

$$R_{CMS}(\textit{ChineseIP}, \textit{Admin}, \textit{Access}) \rightarrow \textit{deny}$$

$$R_{DB}(\textit{CMSRole}, \textit{CMSSchema}, \textit{Write}) \rightarrow \textit{accept}$$

$$R_{FW}(0.0.0.0, \textit{DBServer}, \textit{Send/receive},) \rightarrow \textit{accept}$$

This last example concerns the three subsystems in our IS. A rule in the CMS forbids the access to the admin pages to any user located in China as identified by its IP address. However, the user the CMS uses to connect to the database has

access for modification to the CMS database backend as stated by the second rule. Moreover, the third rule, that belongs to the firewall systems, allows to connect to the database to users in any location. Combining these three rules, a user located in China may be able to access the admin information on the CMS through the database backend.

From the examples, we can conclude that Access-control policies can not be regarded as isolated when they belong to systems situated in different architecture layers. Analysing the access control rules of a component for the absence of anomalies that may introduce unexpected behaviours, requires information from the AC policies of other components it depends on. However, this comprehensive analysis is hampered by two factors: 1) dependencies between components are not explicit, and thus they must be discovered/gathered by hand (complex and error prone in complex environments with many interacting components) 2) the AC information may be represented following a different AC model and stored in different technical spaces requiring domain experts for its analysis. Thus, and integration process allowing for the different policies to be analysed together as components of a complex system is required.

8.3 Approach

In order to tackle the problems discussed above, we propose a model-driven approach that integrates all policies collaborating in the enforcement of access-control in a single model. Our approach requires three steps (see Figure 8.2):

0. **Policy recovery.** AC policies are implemented in concrete systems using a diverse set of mechanisms, often low level and proprietary, like ad-hoc rule languages, specific database dictionaries, etc. As a previous step for our approach we require the policies of each component to be represented in the form of abstract models, from where the complexity arising from the specificities of a given vendor or implementation technology is eliminated and only the AC information is present. This requirement is met by several previous work that investigate the recovery of access-control policies from diverse components. This step is covered by Chapters 5, 6 and 7.
1. **Policy Translation.** Taking as input the models described in the preliminary step, our approach proposes to translate all the policies to the same policy language. This step includes the description of extensions of the target language to make it able to represent component-specific information.
2. **Policy Integration** With all the models translated to the same language, the

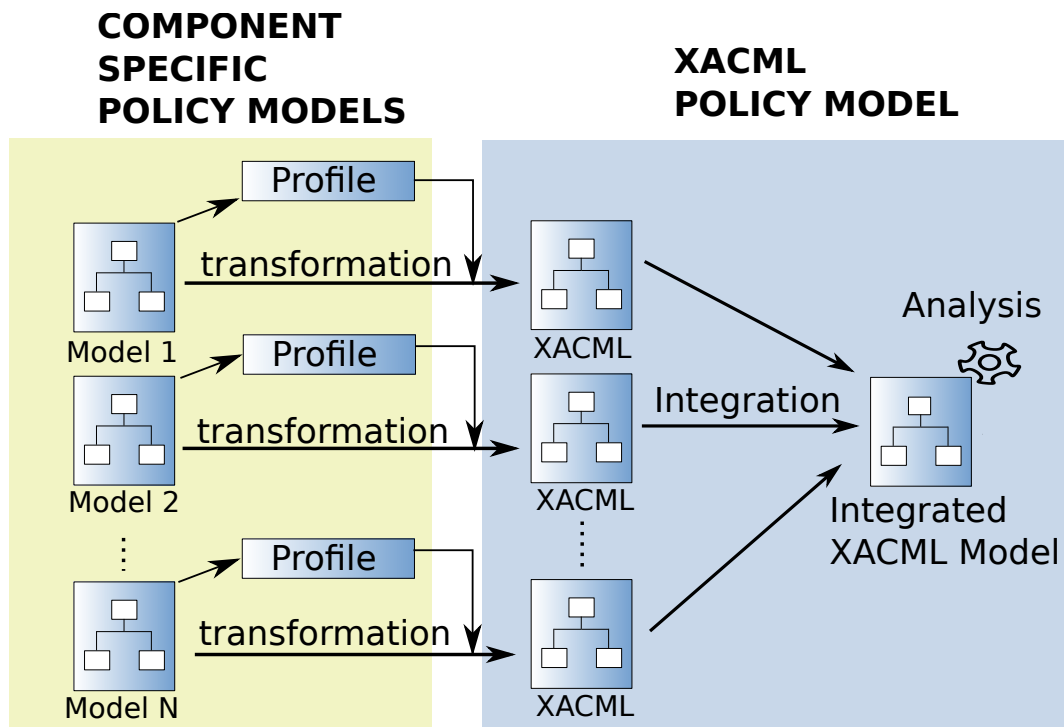


Figure 8.2: Policy Integration Approach

next step is to integrate them all in a single model, along with the dependencies between them. This step requires the discovery of such dependencies, normally implicit.

3. **Policy Analysis Support.** Having all the policies represented in the same model and the dependencies between them made explicit enables the definition of complex analysis tasks. Prior to that, the definition of a number of primitives taking advantage of the model organization is required to ease the building of those analysis tasks. This fourth step is meant to provide that set of primitives.

The following sections are devoted to a detailed description of the steps one to three.

8.4 Policy Translation

All the policies in the IS, potentially conforming to different access-control models and containing domain specific information need to be translated into the same language as a previous step for the integration in a single policy. In order to do so, first, we need to chose a policy language able to represent policies following different policy models and to represent multiple policies in the same artifact.

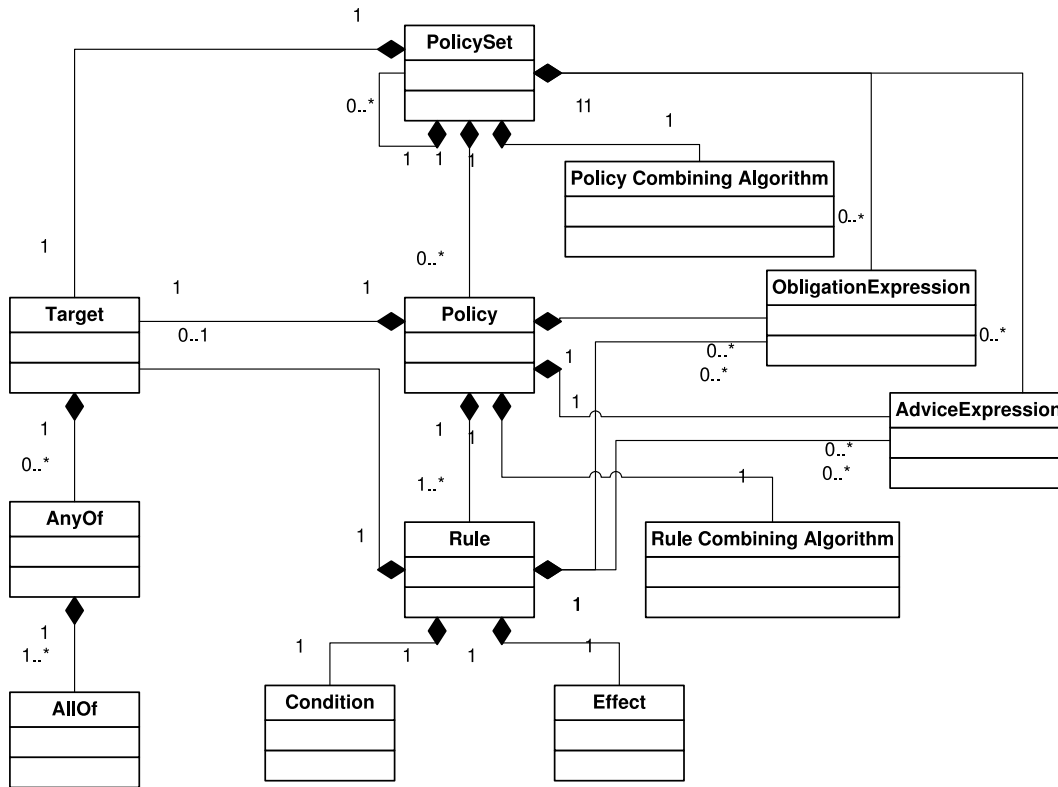


Figure 8.3: XACML Policy Model[59]

8.4.1 XACML Policy Language

XACML[59] is an access control policy language and framework fulfilling these requirements. It follows the Attribute-based access-control model (ABAC)[99]. ABAC along with its extensibility, provides to XACML enough flexibility to represent policies following different access-control models. Other approaches [68, 96, 61] describe languages and tools able to produce flexible access-control models. However, several reasons inclined us to choose XACML. First of all, following the ABAC model, it is able to represent a wider range of security policies (see [50] for the capabilities of ABAC to cover other AC models). While other extensible languages like SecureUML will impose the use of RBAC. Secondly, the standard character of the language, that facilitates its adoption, and, finally, its increasing popularity in both academic and industrial world, that assures its durability and future development. In Figure 8.3 the XACML policy language is depicted.

XACML policies are composed by three main elements *PolicySet*, *Policy* and *Rule*. A *PolicySet* can contain other *PolicySets* or a single *Policy* that is a container of *Rules* (*Policy* and *PolicySet* also specify a rule-combining algorithm, in order to solve conflicts between their contained elements). These three elements can specify a *Target* that establishes its applicability, i.e., to which combination of *Subject*, *Resource* and *Action* the *PolicySet*, *Policy* and *Rule* applies. *Subject*, *Resource* and

Action identifies subjects accessing given resources to perform actions. These elements can hold *Attribute* elements, that represent additional characteristics (e.g., the role of the subject). Optionally, a *Rule* element can hold a *Condition* that represents a boolean condition over a subject resource or action. Upon an access request, these elements are used to get an answer of the type: permit, deny or not applicable.

8.4.2 Translation to XACML and Profiles

Our goal is to translate all the existing policies of the system in hand to XACML policies. However, the component-specific models (see Figures 7.1, 6.1 and 5.3) will typically represent the access-control information in a component-specific way, i.e., they will include concepts of the domain for easing the comprehension and elaboration of policies by domain experts. Those concepts should be preserved in order to keep the expressivity of the policy. For that purpose, XACML profiles need to be defined. These profiles will basically specialize the core concepts of the XACML policy language. In general, a profile will contribute new attributes specializing the concepts of Subject, Resource and Action although specializing other concepts may be necessary mostly when the profile needs to reflect some special feature of the original policy model (take as an example the XACML RBAC Profile [60], where the concepts of PolicySet and Police are extended as well as describing how to arrange these elements in a specific way to achieve the desired goal).

8.4.3 Profiles definition

In order to demonstrate the process of defining a XACML profile, in the following, we describe the development of a XACML profile for the domain of relational database management systems (RDBMSs). The concepts of the domain are extracted from our security database metamodel described in Chapter 6.

First of all, note that the domain of relational databases, and thus our rdbms security metamodel, usually relies on a RBAC model, what should be represented in the profile. There exists already a XACML profile or RBAC. Therefore, our profile will complement the use of this profile by contributing domain specific attributes for Subject, Resource and Action.

We start by defining the profile identifier that shall be used when an identifier in the form of a URI is required:

```
urn:oasis:names:tc:xacml:3.0:rdbms
```

Regarding the Resources, we will describe the following attributes.

```
urn:oasis:names:tc:xacml:3.0:rdbms:resource:database
urn:oasis:names:tc:xacml:3.0:rdbms:resource:schema
```

```
urn:oasis:names:tc:xacml:3.0:rdbms:resource:table
urn:oasis:names:tc:xacml:3.0:rdbms:resource:column
urn:oasis:names:tc:xacml:3.0:rdbms:resource:view
urn:oasis:names:tc:xacml:3.0:rdbms:resource:procedure
urn:oasis:names:tc:xacml:3.0:rdbms:resource:trigger
```

All these attributes are identified by name, as a consequence, their type will be string¹.

```
http://www.w3.org/2001/XMLSchema#string
```

As for the actions, the contributed attribute identifiers listed in the following listing, being all of type string.

```
urn:oasis:names:tc:xacml:3.0:rdbms:action:tableOpt:insert
urn:oasis:names:tc:xacml:3.0:rdbms:action:tableOpt:delete
urn:oasis:names:tc:xacml:3.0:rdbms:action:tableOpt:select
urn:oasis:names:tc:xacml:3.0:rdbms:action:tableOpt:update

urn:oasis:names:tc:xacml:3.0:rdbms:action:dbOpt:alter
urn:oasis:names:tc:xacml:3.0:rdbms:action:dbOpt:drop
urn:oasis:names:tc:xacml:3.0:rdbms:action:dbOpt:create

urn:oasis:names:tc:xacml:3.0:rdbms:action:permissionOpt:grant
urn:oasis:names:tc:xacml:3.0:rdbms:action:permissionOpt:revoke

urn:oasis:names:tc:xacml:3.0:rdbms:action:sessionOpt:set
urn:oasis:names:tc:xacml:3.0:rdbms:action:sessionOpt:connect

urn:oasis:names:tc:xacml:3.0:rdbms:action:codeOpt:execute
```

Finally, and regarding the subjects, the concept of role is already included in the RBAC profile. We will only add an attribute to identify the database elements owned by a subject, as this attribute influences the permissions (commonly, in RDBMS, the owner of a resource has all the permissions and moreover, is allowed to delegate those permissions to others).

```
urn:oasis:names:tc:xacml:3.0:rdbms:subject:owner
```

This attribute is meant to be included in the requests for permissions and its type is boolean.

```
http://www.w3.org/2001/XMLSchema#boolean
```

1. <http://www.w3.org/2001/XMLSchema-datatypes>

XACML	WCMS PIM Metamodel
Policy & PolicySet	A PolicySet and Policy containing the rules described by the CMS model and following the organization prescribed by the XACML RBAC profile
Rule	Each Permission
Subject	Subjects (identifies ans in the XACML RBAC profile) in the CMSs with the given permission granted
Resource	Contents in the WCMS metamodel. Attribute identifiers are created for each kind of content (Node, Page, Post and Comment)
Action	Operations in the WCMS metamodel. Attribute identifiers are created for each kind of operation (AdministrativeOperation, Create, Read, Edit, Delete, Search, Publish, Unpublish).
Condition	Constraints element in the WCMS metamodel.

Table 8.1: WCMS to XACML Mappings

8.4.4 Transformations

Once the profile is available, it is time to create a transformation between the model recovered from the system and XACML+Profiles. Note that to reflect the access-control model used in the RDBMS, we have to explicitly create a rule that in RDBMS is implicit, i.e., the owner has all the rights on the owned element (see Listing 8.1 for an excerpt of the transformation).

The definition of any other profile and corresponding transformations will follow a similar process. In the case of the information system example in Section 8.2, the profiles for the CMS and the network system are easily defined following the same steps. Concretely, for the CMS we will define attributes extending the core concepts of XACML following the types defined in Chapter 7 and then combining its use with the use of the RBAC profile. In Table 8.1 we describe the mappings and attribute identifiers needed for transforming and WCMS model to XACML.

As for the firewalls, several mappings to use as a basis for the profile exists, including the use of roles [38] or not [77]. For simplicity, we extend the latter (see Table 8.2) to include domain concepts (as host, zone, protocol, etc), discarding the discovery/creation of implicit roles.

8.5 Integration

Once we have all the policies represented in the same policy language, the next step is to organize the policies within a global model representing all the access-control policies in the IS. Key within this step is to unveil the implicit dependencies between policies situated at different architecture levels to make them explicit. First

XACML	Filter PIM Metamodel
PolicySet	A PolicySet containing a Policy is created for each firewall in the PIM
Policy	All the Connections and Exceptions belonging to a given firewall
Rule	A single connection or Exception
Subject	Source NetworkElement address and source port of a given Connection or Exception
Resource	Target NetworkElement address and target port a given Connection or Exception
Action	Not mapped. The action is always the ability of sending a message.
Condition	Protocol field

Table 8.2: Network to XACML Mappings

of all, we need to decide which structure to use in order to represent the policies and their dependencies in a single XACML resource. The policy of each component, translated to XACML will be stored in a PolicySet, so that we can use the PolicySetIdRef to link it to other policies in the system. Note that some scenarios will require the policy of the component to be split in several PolicySet and Policy elements as is the case when using the RBAC profile. For simplicity, in the rest of the chapter we will consider the policy of a component as the element containing its rules, disregarding how they are organized using XACML elements. Note also that the proposed structure is not intended to be deployed as is in a XACML framework but to enable analysis capabilities. Component policies must be deployed individually.

Starting from the individual policies, we need a process to discover the dependencies between them, so that the references can be properly set. We propose here a process based on exploiting context information understanding it as context environmental information that helps to relate the system in hand to other systems, e.g., a database user in a CMS or its IP address.

We consider this context information relevant not only to unveil the dependencies but also for the analysis of the system. Therefore, it needs to be stored along with the policy representation in order to have it available when needed. XACML does not provide a specific place to store this kind of information in the policy language. To overcome this limitation and to have the least possible impact we add this information in the description field of the PolicySet element.

```
<xs:complexType name="PolicySetType">
  <xs:sequence>
```

```

        <xs:element ref="xacml:Description" minOccurs="0"/>
        ...

<xs:element name="Description" type="xs:string"/>

```

In this field we store a string representing a key, value map with the corresponding environment values for the Policy or PolicySet. This is shown in the example below.

```
Context { dbUserName:anonym; IPAddress:192.000.111.0 }
```

With the context information available, the process to find the dependencies between policies is described in Algorithm 1. Basically, the algorithm works as follows:

For each context parameter in a given policy it searches if there is any rule using that attribute value in any of the other policies. If this is the case, a dependency exists between both policies and as such is registered. Note that the algorithm has been optimized by considering that no circular dependencies exist (the set of candidate policies, initialized to all the policies in line 3, gets modified in line 13 to achieve this behaviour). This assumption stems from the nature of multilayer ISs where normally upper components depend only on components in lower layers. This optimization can be dropped if necessary for other scenarios.

Algorithm 1

```

1:  $P \leftarrow$  All Policies
2: for each  $P_i \in P$  do
3:    $Dependency[P_i] \leftarrow \emptyset$ 
4:    $Candidates[P_i] \leftarrow P$ 
5: end for
6: for each  $P_i \in P$  do
7:    $C \leftarrow$  All Context Attributes in  $P_i$ 
8:   for each  $C_i \in C$  do
9:     for each  $P_j \in Candidates[P_i]$  do
10:       $A \leftarrow$  All Rule Attributes in  $P_j$ 
11:      if  $C_i$  in  $A$  then
12:         $Dependency[P_i] \leftarrow Dependency[P_i] \cup \{P_j\}$ 
13:         $Candidates[P_j] \leftarrow Candidates[P_j] \setminus \{P_i\}$ 
14:      end if
15:    end for
16:  end for
17: end for

```

Figure 8.4 shows the result of applying our approach to our IS example. A *policySet* element has been created for each of the system components: firewall,

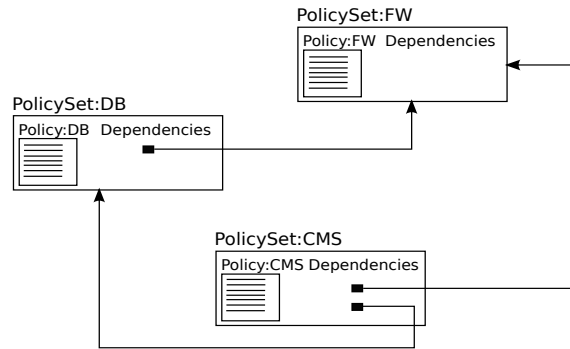


Figure 8.4: Policy Organization

database and CMS. These *policySets* contain the translated to XACML access-control *policy* of each component along with references to its dependencies as calculated by algorithm 1. Considering the following set of context attributes for each component:

```

//DB context
Context { IpAddress : 111.222.1.10 }
//CMS context
Context { dbUserName : anonym; IpAddress : 111.222.1.12 }
//Firewall context
Context { }
  
```

the results is that the database *policySet* holds a dependency on the firewall *policySet* (due to the IP address context attribute) while the CMS *policySet* holds dependencies to the firewall and de database *policySets* (due to the database user and IP address context attributes).

8.6 Policy Analysis Support

Representing all the policies collaborating in enforcing access-control in a single model, along with making explicit the dependencies between policies situated in different layers, enables the possibility of performing analysis and manipulation tasks unavailable when focusing on individual policies. Beyond a number of common operations, such analysis task differ greatly, thus, here we focus in one specially critical analysis task, the detection of inter-component anomalies. Prior to that, we will present a number of basic operations introduced with the purpose of easing the manipulation of our integrated model.

8.6.1 Basic Operations

Our model can be easily queried to extract useful information by using the OCL standard query language. However, there is a set of operations that will be com-

Operation	Description
<code>getDependents(p:Policy) : Sequence{Policy}</code>	Given a policy P , returns the sequence of policies having this policy as dependency.
<code>getDependencies(p:Policy) : Sequence{Dependency}</code>	Given a policy P , returns the sequence of direct dependencies.
<code>getAllDependencies(p:Policy) : Sequence{Dependency}</code>	Given a policy P , returns the sequence of ALL the dependencies, direct and indirect.
<code>resolveDependency(d:Dependency) : Policy</code>	Given a dependency D , returns its target Policy P .
<code>getDependencySource(d:Dependency) : Policy</code>	Given a dependency D , returns its source Policy P .
<code>getContextAttributes(p:Policy) : Sequence{ Tuple {key:String,value:String} }</code>	Given a Policy P , returns a sequence of tuples{key:String,Value:String}, representing the context attributes
<code>getRelevantRules(p:Policy,p2:Policy) : Sequence{Rule}</code>	Given two policies, P_i and P_j , with P_i dependent on P_j , returns the rules in P_j related to the context attributes of P_i , i.e., the set of rules of P_j P_i depends on

Table 8.3: OCL Operations

monly used and as such, it is worth defining and implementing as a reusable library. In that sense, we present here a list of operations implemented with OCL that are basic for working with our model.

Table 8.3 presents a description of this set of basic operations. Basically, we present operations to work with the dependencies, *getDependants*, *getDependencies*, *getAllDependencies* *resolveDependency* and *getDependencySource*; operations to obtain the context attributes of a policy, *getContextAttributes*; and operations to obtain the rules related with context attributes in a dependency relation, *getRelevantRules*.

8.6.2 Instantiating policy similarity

One important analysis task is the detection of anomalies that appear when several policies work together. The problems these anomalies cause vary from simple incrementing the complexity of policies to the introduction of unexpected behaviour of a component w.r.t. to its defined policy.

In order to be able to detect these anomalies we need to compare rules. This comparison is done for the purpose of checking if 1) Conditions hold for the same set or different set of values 2) The rule effects when the conditions hold are conflicting or not. This process, which we call rule similarity evaluation following

the terminology in [65], can be performed following different approaches. Here, due to the relative simplicity of the syntactical analysis they propose, we adapt the approach proposed in [65] to the case of policies in different architectural layers. Other approaches could however be also adapted to our specific case. Note that we discard to perform a generic policy similarity process. Being in different architectural layers, the policies will be mostly disjoint, with only some overlapping points. Thus, calculating rule similarity yields enough information.

Therefore, we adapt the approach in [65] in the following way:

1. We restrict the space of comparison between policies to only the relevant rules as calculated from the dependencies (cf. Section 8.3). Any given rule will depend only on those rules related to the attributes used to calculate the policy dependencies so is this concrete subset the one that needs to be checked in order to detect possible anomalies.
2. We include dependency information during rule similarity calculation, so that extra information regarding the components involved is available.
3. We instantiate the Rule similarity types for the case of rule situated in different architectural layers, one depending on the other. Subsection 8.6.2 details this instantiation.

Rule Similarity Types

From [65], the possible values to obtain when calculating the similarity between two rules w.r.t. a given attribute are in the tuple:

{Converges, Diverges, Restricts, Extends, Shuffles}.

Those values have the following definition:

Converge: Two rules 'converge' if the sets of values are equal with respect to which their conditions hold.

Diverge: Two rules 'diverge' if the sets of values do not intersect with respect to which of their conditions hold.

Restrict and extend: A rule 'restricts' (or 'extends') another rule if the sets of values with respect to which its conditions hold contain (or is contained in) the set of values computed for the other rule.

Shuffle: Two rules 'shuffle' if the sets of values for which their conditions hold intersect, but no one is contained in the other.

When rules belong to policies situated in different architecture layers and dependencies between them exist, the aforementioned values can be instantiated to give hints about the possible presence of anomalies. Focusing on the undesired effects these anomalies may produce and considering r_i depending on r_j , we propose to instantiate the similarity values to four values, described as follow:

- **Security Risk:** The combination of r_i and r_j may cause a security hole. This happens when r_j allows requests for values r_i does not allow. We consider the risk partial when r_j only allows some of the r_i denied values. Example 2 in Section 8.2 belongs to this category, as the network layer, combined with the database layer, allows request the CMS does not.
- **Service Risk:** The combination of r_i and r_j may cause the component to which r_i provides access-control not to be able to provide the expected service. This happens when r_j denies requests for values r_i allows. We consider the risk partial when r_j only denies some of the r_i allowed values. Example 1 in Section 8.2 shows a partial service risk.
- **Redundancy:** r_i or r_j may be eliminated without impact to the behaviour in the system. This may happen when both rules deny requests for the same values. Policies containing those rules may be refactored to reduce complexity.
- **No Risk:** The combination of r_i and r_j does not generate any risk.

The assignment of these values to the original rule similarity types depends on the effects (deny, accept) of the rules under analysis. Table 8.4 shows the assignment for all the possible combination of effects.

We would like to notice here that the actual presence of anomalies between two rules whose similarity hints to it, depends on the nature of the involved systems and how they interact between them.

Rule Similarity Calculation

Algorithm 2 describes the process of calculating the similarity of rules over our infrastructure given a rule and an attribute to check.

Basically, the algorithm iterates over the policies the policy containing the rule depends on, retrieving relevant rules (lines 9 and 13) and calculating the similarity

value (line 18) to produce an anomaly report (line 19). It is important to note that when the dependency is indirect, i.e., the dependency relationship is established through another policy, we need to get the relevant rules w.r.t. this latter policy having the direct dependency (line 11 to 15). This is specially important because a given policy may have both, a direct and an indirect dependency with another policy, each one yielding a different set of relevant rules. As this information is relevant for performing further analysis, each rule is tagged with its dependent policy (lines 10 and 14).

Let us take a look of how the similarity is calculated for the examples presented in Section 8.2.

Regarding the first example, we want to know if given the database rule and the time attribute there exists any anomaly:

$$R_{DB}(RoleX, TableX, Write, 8:00 - 16:00) \rightarrow accept$$

Following the proposed algorithm, the policy dependencies are retrieved, that in this case consists only in a dependency towards the firewall policy. Using the context attributes, the rules in the firewall policy related to the database are retrieved. Finally, from this set of rules, the ones containing the time attribute are checked for similarity with the database rule and tagged in consequence. We can then show only those ones having a similarity implying an anomaly. In that subset we will have the second rule in the example, as it uses a context attribute, the time attribute and the calculated similarity has the value of restrict, which may cause an anomaly as shown in the Table 8.4.

$$R_{FW}(Local, DBServer, Send/receive, 8:00 - 14:00) \rightarrow accept$$

As for the second example, the process starts in a similar way, by retrieving the dependencies of the CMS policy containing the rule and attribute to be checked, in this case, the source IP address.

$$R_{CMS}(ChineseIP, Admin, Access) \rightarrow deny$$

However, now we will have two kinds of dependencies. The CMS policy depends directly on the database and firewall policies, but it also holds an indirect dependency to the firewall policy through the database one. Thus, three sets of rules are retrieved, those of the firewall and database policies related to the CMS context attributes (IP address and database user) and those of the firewall related with the context attributes of the database (IP address of the server).

The policy similarity calculated on the set of retrieved rules we will exhibit not only the possible anomalies the policy of the CMS has with respect to the database and the firewall directly, but also those anomalies arising from the combination of the effects of rules in the database and firewall together. Thus, among other possible anomalies present in the firewall or database configuration we will retrieve the one associated with the following rule:

$$R_{FW}(0.0.0.0, DBServer, Send/receive,) \rightarrow accept$$

It gives access to the database server (back-end of the CMS) to users in a location forbidden by the CMS policy. This rule retrieved from the database dependency and tagged that way, informs us about an anomaly (shadowing) between the CMS and the firewall involving the database system. The security expert will only need to retrieve the database relevant rules w.r.t. the CMS to have a complete picture of the problem.

$$R_{CMS}(ChineseIP, Admin, Access) \rightarrow deny$$

$$R_{DB}(CMSRole, CMSSchema, Write) \rightarrow accept$$

$$R_{FW}(0.0.0.0, DBServer, Send/receive,) \rightarrow accept$$

Obtaining this information will not have been possible without the integration of the policies and the discovery of their dependencies.

8.7 Tool support

In order to validate the feasibility of our approach, a proof-of-concept prototype implementation has been developed under the Eclipse environment by using Model-driven tools and techniques. Concretely, our implementation is based on two features:

Model Representation. Our approach takes as input domain-specific access-control models extracted from different components in order to translate them to XACML models. To be able to do that, a XACML policy metamodel (models conform to metamodels, which define the main concepts and relationships of the domain) is required, so that models conforming to it can be created. We have used, EMF[92], the de-facto modeling framework standard for that purpose.

Providing the XACML XSD schema² as an input to EMF, the framework allowed us to generate the XACML policy metamodel, and in turn, to generate java

2. <http://docs.oasis-open.org/xacml/3.0/XSD/cs-xacml-schema-policy-01.xsd>

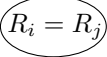
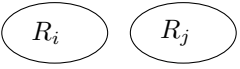
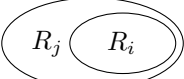

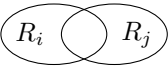
Rule type	similarity	Authorized requests	$R_i^{Accept}, R_j^{Accept}$	R_i^{Deny}, R_j^{Deny}	R_i^{Deny}, R_j^{Accept}	R_i^{Accept}, R_j^{Deny}
R_i Converges R_j			No Risk	Redundancy	Security Risk	Service Risk
R_i Diverges R_j			Service Risk	No Risk	No Risk	No Risk
R_i Restricts R_j			No Risk	No Risk	Security Risk	Service Risk
R_i Extends R_j			Service Risk (Partial)	No Risk	Security Risk (Partial)	Service Risk (Partial)
R_i Shuffles R_j			Service Risk (Partial)	No Risk	Security Risk (Partial)	Service Risk (Partial)

Table 8.4: Policy rule similarity type instantiated

code plugins for the manipulation of model instances, including a tree-based editor. Note that these models instances can be, in turn, serialized using a XML syntax.

XACML identifiers, datatypes, etc. are integrated in a similar way i.e., by extracting a metamodel through EMF and linking it to the XACML metamodel.

Model Query&Transformations.

Listing 8.1: Database model 2 XACML

```

1 module DB2XACML;
2 create OUT : XACML from IN : SecurityDB, IN2:DBProfile, IN3:RBACProfile;
3 rule createdBPolicySet{
4   from
5     root: DBSecurity!Database
6   to
7     policySet : XACML!PolicySetType (
8       policySetId <- 'DatabasePolicy',
9       policySet <- root.subjects
10      ->collect(e | thisModule.CreateRolePolicySet(e))
11      ->append(root.subjects
12      ->collect(e | thisModule.CreatePermissionPolicySet(e)))
13 }
14 lazy rule CreateRolePolicySet{
15   from
16     dbRole : DBSecurity!Role
17   to
18     rolePolicySet: XACML!PolicySetType (
19       policySetId <- dbRole.name + ':role',

```

Algorithm 2 Calculate similarity

```

1:  $r \leftarrow \text{Initialrule}$ 
2:  $a \leftarrow \text{Initialattribute}$ 
3:  $P \leftarrow P/r \in P$ 
4:  $D \leftarrow \text{getAllDependencies}(P)$ 
5:  $S \leftarrow \text{getDependencies}(P)$ 
6: for each  $D_i \in D$  do
7:    $P_i \leftarrow \text{resolveDependency}(D_i)$ 
8:   if  $D_i$  in  $S$  then
9:      $R \leftarrow \text{getRelevantRules}(P, P_i)$ 
10:     $\text{tagRules}(R, P)$ 
11:   else
12:      $P_j \leftarrow \text{getDependencySource}(D_i)$ 
13:      $R \leftarrow \text{getRelevantRules}(P_j, P_i)$ 
14:      $\text{tagRules}(R, P_j)$ 
15:   end if
16:   for each  $r_i \in R$  do
17:     if  $a \in r_i$  then
18:        $\text{sim} \leftarrow \text{calculateSimilarity}(r_i, r, a)$ 
19:        $\text{reportAnomalyCheck}(\text{sim}, r_i, r, )$ 
20:     end if
21:   end for
22: end for

```

```

20     policySetIdReference <- dbRole.name + ':permission')
21 }
22 lazy rule CreatePermissionPolicySet{
23   from
24     dbRole : DBSecurity!Role
25   to
26     permissionPolicySet: XACML!PolicySetType (
27       policySetId <- dbRole.name + ':permission',
28       policySet <- Sequence{permissionPolicy}
29     ),
30     permissionPolicy: XACML!PolicyType (
31       policyId <- 'Permission for' + dbRole.name,
32       "rule" <- dbRole.grantedPrivileges
33       ->collect(e | thisModule.CreateRoleRules())
34   )
35 lazy rule CreateRoleRules{
36   from
37     dbPermission: DBSecurity!Permission
38   to
39     xacmlRule: XACML!RuleType
40     ...
41 }

```

Listing 8.2: getAllDependencies Helper

```

1 helper context XACML!PolicySet def:getDependencies
2 : Sequence(TupleType(P:String, D:String)) =
3   self.policySetIdReference->collect
4   (e | Tuple{P = self.policySetId, d = e});
5
6 helper context String def:resolvePolicySet

```

```

7 : XACML!PolicySet =
8   XACML!PolicySetType.allInstances()
9   ->select(e | e.policySetId = self)->first();
10
11 helper context XACML!PolicySet def:getAllDeps
12 : Sequence(TupleType(P:String , D:String)) =
13   self.getDependencies->append(
14   self.getDependencies->collect(e | e.d.resolvePolicySet.getAllDependencies)->
     ↪flatten())->flatten().asSet();

```

Once XACML models are available, we can perform the transformations from the component-specific models to XACML and the operations and algorithms described in Sections 8.5 and 8.6. We have used the ATL model-to-model transformation language for that purpose. Concretely, the following ATL transformation have been created:

1. A model transformation for each component model to transform it into a XACML model.
2. A library of helpers, an ATL mechanism to factorize OCL operations, representing the basic operations in section 8.6.
3. A model transformation for the integration of all the XACML models following the algorithm in 1.
4. A model query for the detection of anomalies, following the algorithm 2.

Listings 8.1 and 8.2 show examples of such ATL implementations. The former is an excerpt of the Database security to XACML transformation and the latter a Helper implementation of the *getAllDependencies* operation.

Related Work

9.1 Network AC Reverse-engineering

Several other works tackle the problem of extracting access control policies from network configurations but they either are limited to analyzing one single firewall component or focus on a specific analysis task and thus they do not generate a usable representation of the firewall/s under analysis. Moreover, these latter works require as an additional input the network topology, instead we are able to calculate it as part of the process.

More specifically, [94] proposes a technique that aims to infer the higher-level security policy rules from the rules on firewalls by extracting classes (types) of services, hosts and protocols.

To do so, first rules are flattened, i.e., the set of rules is reorganized in order to make individual rules independent of the order. As stated in that work, this can be achieved by following very simple approach approach:

Given a set of rules R , where R_i has higher priority than $R_j \iff i < j$ then a flattened rule set can be obtained by simply replacing R_j by $(\bigwedge_{i=1}^{j-1} \neg R_i) \wedge R_j, 2 \leq j \leq n$ where n is the total number of rules in the set. However, using such an approach leads to an explosion in the number of rules that increases the complexity of the policy. The authors deal with this explosion by proposing a graph based approach (rules to be generated are paths on the graphs towards and allow or deny node) that reduces this rule explosion. Once rules are *flattened*, its number is reduced by grouping hosts, services and protocols into classes and merging rules containing same classes of objects.

Transforming a rule set to a flattened rule set is interesting when dealing with

policies mixing positive and negative logic (i.e., policies containing both, accept and deny rules). In that sense, our network metamodel and our algorithm to fill it by classifying rules in *connections* and *exceptions* achieves a similar result with less effort and avoiding any kind of rule explosions. Effectively, after applying our algorithm, *connections* do not depend of the order while *exceptions* only need to keep the order inside their *connection*. This classification process also reduces the need for the grouping phase in [94] as, first, the number of rules does not change w.r.t the original configuration, and second, rules are grouped around parameters when effects (accept, deny) differ. Further grouping, filtering, etc. can be performed by using model transformations as we have showed in Chapter 5 for the calculation of global relevant rules and the derivation of the topology.

Our example in Chapter 5 (see 5.2), when translated to our exception oriented model will consist in two connections (independent of the ordering) containing two exceptions each. This makes a total of 6 rules as in the original file (disregarding the rules setting the global policy to open or close.). Applying the method in [94] these six rules are transformed to a graph with many more paths. Note also that this graph can no be pruned like in their example to eliminate the deny node, as it will require transforming a policy mixing positive and negative logic to one using only one of them, what leads to overcomplicated (because of the number of rules) unnatural rule sets. We want also to notice that their approach works only for one firewall and is specially tailored to the Netfilter Iptables firewall.

In [64] a method and tool to discover and test a network security policy is proposed. The configuration files along with the description of the network topology are used to build an internal representation of the policy that can be verified by the user through queries in ad-hoc languages. The tool is aimed at answering question of the form *does the policy allow service s from a to b?*.

In this sense, this tool is equivalent to our approach although it present some drawbacks and limitations.

First of all, their tool, FANG, requires the user to provide as an input a detailed model of the topology of the network where all elements enforcing filtering rules along with their interfaces need to be specified. Conversely, our approach works only with the configuration files and is able to derive a representation of the topology of the network as seen in Chapter 5 Section 5.3.

Regarding the queries, our model supports the same kind of querying although they are performed in quite a different way. While our approach uses an standard model query language (OCL) over an EMF model, FANG actually simulates the flux of packages through the network. OCL, being standard is widely known and has been deeply study by the research community. As a consequence, queries in OCL can be preformed in a very efficient way (lazy and incremental OCL implementations have been contributed). Listings 9.1 and 9.2 show a query written in

OCL and the FANG system respectively. While they are equivalent, the flexibility of the FANG's query is limited to the use of wildcards while OCL is a powerful query language allowing us to perform many different kinds of queries and metrics.

Listing 9.1: OCL query: Can a given host send HTTP requests to the server?

Evaluating:

```
self.connections->exists(e | e.source.ipAddr='111.222.2.54' and
                          e.target.ipAddr='111.222.1.17' and
                          e.srcPort = '80')
```

Results:

```
false
```

Listing 9.2: Fang query: Can a given host send HTTP requests to the server?

```
struct query {
    struct hostgrp *src;
    struct hostgrp *dst;
    struct servicegrp *service;
};
```

Instantiated to:

```
query(111.222.2.54, 111.222.1.17, 80);
```

The FANG queries work over an internal model representation, independent from the concrete firewall vendor but representing the rules in the same abstraction level. This internal model is not available to the user, and thus it can no be further manipulated or analyzed. This will prevent the network models to be combined with models in other subsystems, hampering the global analysis of the AC implementation of a given information system.

In [69] a similar approach is described. The authors use the Margrave tool for firewall analysis. They model the firewall rules by using first-order logic. Then they provide the means of querying the build model by using an ad-hoc (internally, queries are also translated to first order logic) querying system and language. Although the query language is more flexible than the one in [64], it suffers from the fact of being tool-specific. This obliges the user to learn the syntax and semantics of the language.

In Listing 9.3 a Margrave query is shown. This query answers the following question: What rules deny a connection from the manager's PC to port 80 somewhere outside our network other than the blacklisted host. As we can see the query syntax is quite complex and requires still low-level and very specific knowledge like the firewall interfaces, topology, etc. Our model abstracts from the topology, showing only connections and exceptions. This way, it can be queried without any knowledge of the underlying topology.

Listing 9.3: Margrave Query

```

EXPLORE prot-TCP = protocol AND
192.168.1.2 = fw1-src-addr-in AND
in_lan = fw1-entry-interface AND
out_dmz = fw2-entry-interface AND
hostname-int = fw1 AND
hostname-ext = fw2 AND

fw1-dest-addr-in IN 10.200.0.0/255.255.0.0
NOT 10.200.200.200 = fw1-dest-addr-in AND
port-80 = fw1-dest-port-in AND

internal-result(<reqfull-1>) AND

(NOT passes-firewall(<reqpol-1>) OR
internal-result(<reqfull-2>) AND
NOT passes-firewall(<reqpol-2>))

UNDER InboundACL
INCLUDE

InboundACL:int-in_lan-line-12_applies
(<reqpol-1>),
InboundACL:int-in_lan-line-17_applies
(<reqpol-1>),
InboundACL:ext-out_dmz-line-19_applies
(reqpol-2>),
InboundACL:ext-out_dmz-line-21_applies
(<reqpol-2>),
InboundACL:ext-out_dmz-line-24_applies
(<reqpol-2>)

```

Also, as in [64], the model remains an internal representation aimed at supporting user queries and not intended to be directly used manipulated by the user. Thus, the representation, although independent from the concrete firewall vendor, lies in the same abstraction level as the configuration files. As the focus of this thesis is to extract the abstract access-control model from components, we believe our model representation and the use of standard MDE tools more suitable for this task, being approaches like FANG and Margrave more suitable for low level analysis.

[21] proposes a bi-directional method to enforce and reverse engineer firewall configurations. It promotes the use of an intermediate policy representation but does not provide a model for such representation nor specific processes to perform the enforcement and the discovery tasks.

Some other works provide a metamodel for representing firewall configurations. Nevertheless, a reverse engineering process to populate those models from existing configuration files is not provided, and in our opinion, the abstraction of the models level is still too close to the implementation details, therefore limiting their usability.

Concretely, In [101] the authors present a platform independent language for firewalls in the form of XML (with the syntax specified as a DTD). As we can observe in the example of Listing 9.4, the language remains quite complex and near

to the implementation level. In fact, the language is designed as a pivot language for a code generation tool and it is not supposed to be directly manipulated by users.

Listing 9.4: FWBuilder firewall language

```
<Firewall hostOS="linux24" id="id47505D0516470" name="MyFirewall" platform="
iptables ">
  <Interface dyn="False" id="id47505D0B16470" name=" if0 " unnum="False">
    <IPv4 address=" 192.168.1.1 " id="id47505D0C16470" name=" MyFirewall:if0:ip "
    netmask=" 255.255.255.0 "/>

    <physAddress address="00:17:f2:ea:ee:35" id="id47505D3816470"
    name=" MyFirewall:if0:mac"/ >
  </Interface>
  <Interface dyn="True" id="id47505D0D16470" name=" if1 " unnum="False"/>
  <Interface dyn="False" id="id47505D0F16470" name=" l0 " unnum="False"
  unprotected="False">
    <IPv4 address=" 127.0.0.1 " id="id47505D1016470" name=" MyFirewall:l0:ip "
    netmask=" 255.255.0.0 "/>
  </Interface>

  <Policyid="id47505D0816470">
    <PolicyRule action="Deny" comment="" direction="Both" disabled="False"
    id="id47505ECE16470" position="0">
      <Src neg="False">
        <ObjectRef ref=" sysid0 "/>
      </Src>
      <Dst neg="False">
        <ObjectRef ref="id47505CE816470"/>
      </Dst>
      <Srv neg="False">
        <ServiceRef ref="id47505D0216470"/>
      </Srv>
      <Itf neg="False">
        <ObjectRef ref=" sysid0 "/ >
      </Itf>
      <When neg="False">
        <IntervalRef ref=" sysid2 "/>
      </When>
    </PolicyRule>
  </Policy>
</Firewall>
```

More near to ours, in [81], the authors present a PIM metamodel (see Figure 9.1) for firewalls (They extend the work from the same authors presented in [80]). Contrarily to our approach, their metamodel is intended to be the starting point for the specification of firewall policies, being the actual firewall configuration files generated by a series of model transformation. Although very similar to ours, their metamodel does not provide support for the representation of exceptions (and mis-configurations). Moreover, it is intended to model single firewalls as no mechanisms for the combination of multiple policies is provided (in our metamodel, connection rules from several firewalls can be represented as firewalls itself are represented in the model, moreover, each connection holds an attribute identifying the firewall containing the rule). On the other hand, the metamodel the present supports the concept of NAT, that is missing in ours. However, our metamodel and approach

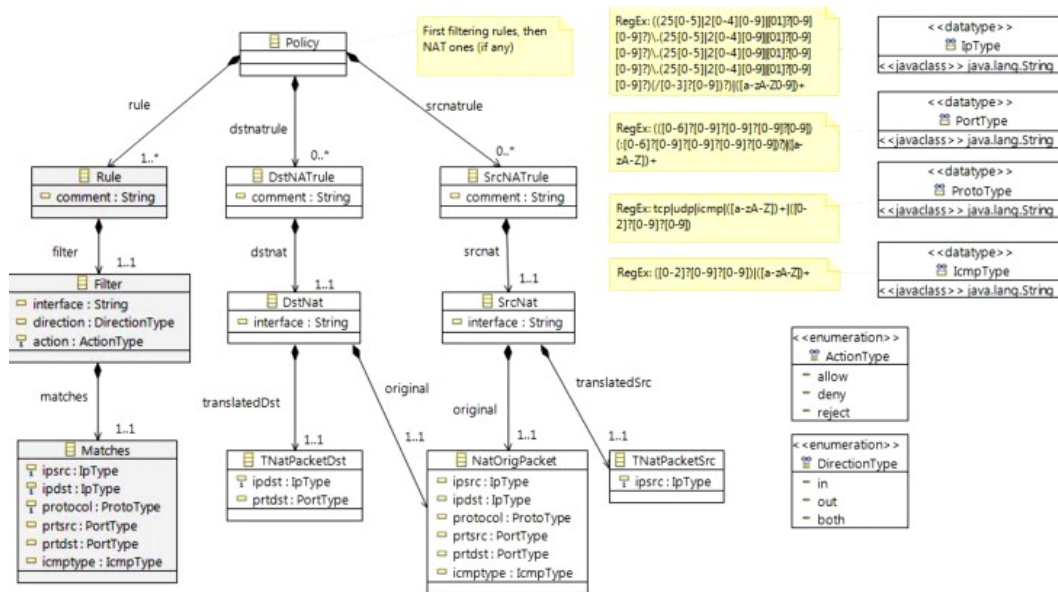


Figure 9.1: CONFIDENT PIM meta-model [81]

could be easily extended to include NAT without requiring any modification of the proposed analysis tasks.

As in [69], several approaches have proposed formal models for the representation of firewall policies. Among them, in [24] the authors present a Higher-order logic model able to represent and analyze firewall policies. They have also extended their model to deal with the stateful case in [25]. All these formal representations, while powerful, are very difficult to use, requiring the user to know not only the domain of network packet filtering, but also the domain of the formalism of choice, thus preventing their adoption. Moreover, their main strength, e.g., the automatic validation and verification of properties, etc, depends in the ability of solvers to explore the domain of solutions, what it is know to only work in a scalable way in bounded search. Nevertheless, our model can be used as a pivot language between the real configuration, the user and these formal models by means of model-transformations (text-to-model, model-to-model and model-to-text).

9.2 RDBMS AC Reverse-engineering

To the best of our knowledge, ours is the first security metamodel tailored to the security mechanisms available in relational databases. Our metamodel integrates RBAC concepts but extends the standard RBAC model to cover the full spectrum of database security mechanisms useful to express access-control policies. Although several (extensions to) modelling languages able to model security concerns[52, 61] have already been proposed, they are aimed to model the security aspects of the whole information system and thus lack of precision to define in detail database-

specific access control policies.

Regarding the reverse-engineering of security aspects, in [89] the authors present an approach to discover and resolve anomalies in MySQL access-control policies by extracting the access-control rules and representing them in the form of Binary Decision Diagrams (BDD). Although they do not describe a method for extracting the access-control information from a running database, they provide a formal model to represent it.

For them, the access-control of a database is composed by rules of the form:

$$R_i : C_i \rightsquigarrow PV_i$$

Where C_i is a set of conditions that the rule must hold to obtain the permissions in the privilege vector PV_i

The conditions are modeled as a boolean conjunction of fields:

$$C_i = f_1 \wedge f_2 \wedge \dots \wedge f_k$$

Privileges are also modeled as a boolean conjunction:

$$PV = P_1 \wedge P_2 \wedge \dots \wedge P_k$$

Then, permissions and grant tables are modelled as BDDs, so that the evaluation of access-control requests is enabled.

As fields they consider *Host IP*, *user name*, *database name* and *column name*.

Our metamodel supports directly all these conditions but the Host IP condition that is supported by the analysis of triggers (we consider context checks are often performed by using fine grained constraint defined in procedural code).

Compared to our metamodel, they notably do not model some key access-control elements. Model views that constitute a case of fine-grained, column-level and content-level access control mechanism widely used in real databases are not taken into account. Support for the representation of roles and the association between roles and privileges and between users and roles is also missing. This limits the applicability of their model representation to databases not supporting RBAC (which is the case of MySQL but not of other important database vendors like Oracle, PostgreSQL and Microsoft SQL Server). Ownership is also ignored. Relational databases usually follow a DAC model, where the owner of an element have total control over the permissions on his objects. Although this information can be flattened and the set of permissions obtained by ownership represented as normal grants we consider that doing so hampers the possibility of analysing the policy as it is by database

experts. Conversely, the provided model seems to be completely focused in the automatic analysis of the semantics of the policy, disregarding policy understanding tasks.

Finally, the contribution of procedural code is also not taken into account. In Chapter 6 we discussed how procedural code items, e.g., triggers and stored procedures, modify the privileges as described in the database dictionary grants table and we provided an approach to integrate information extracted from them into models conforming to our database security metamodel.

Summarizing, compared to us, the approach in [89] do not provide a higher-level, easier to understand and manipulate representation of the extracted policies nor take into account the contribution that several other elements like triggers, stored procedures and views provide to access-control. However, their formal analysis of consistency and absence of errors based on BDDs could be reused as an extra analysis task in our approach. Indeed, a transformation from our metamodel towards an extended version of their formal representation is straightforward and will complement both approaches.

Finally, there exist plenty of reverse engineering efforts [62], [33], [79],[14] (among many others) focused in recovering a (conceptual or logical) schema from a database. Nevertheless, none of them covers security aspects and therefore, they could benefit from our approach to extract a richer model.

9.3 CMS AC Reverse-engineering

Being the popularity of Content Management Systems relatively recent, it has not attired to much attention from the research community. To the best of our knowledge, our is the first metamodel specially tailored to the representation of access-control information for the domain of CMSs.

Some tools for checking the configuration of WCMSs have been provided and analysed by the scientific communities. However, these tools are focused in low-level security aspects like management of cookies or prevention of SQL injection vulnerabilities [66, 97].

Nearer to the extraction of high-level model representations of access-control, approaches for extracting AC information from dynamic web applications source code are presented in [44, 10].

In [44] the authors extract access-control models out the Moodle eLearning framework. They parse the PHP code and obtain a Control Flow Graph where access-control patterns (e.g., calls to the `has_capability` and `require_capability` functions) that is then translated into an automaton for model checking. In [10], a more generic approach is presented. SecureUML models are extracted from dynamic

web applications by applying dynamic and static analysis techniques. This second approach is more similar to *as*, as they provide a high-level abstract model of the system representing the domain entities as extracted from the database and the security information as stereotypes. Nevertheless, the model they provide is not specifically tailored to CMSs. Conversely, the entities are application specific, what hampers the implementation of reusable CMS-specific analysis and migration tasks.

Notice also that those approaches are focused in a low level analysis of the implementation of access-control over web applications. However, as discussed in Chapter 7, current CMS systems include access-control mechanisms and tools to set a policy. Once the policy is defined it is stored in the CMS back-end, normally a database management system. Low level code will thus rely in this back-end to enforce the Access-control policy. Therefore, we believe than in the case of CMSs, extracting the information directly from the policy specification back-end provides enough information as we consider that the code implementation correctly follows the specification and that providing a better representation and understanding of the specification remains a necessary step.

9.4 AC Policy Integration

The integration of security policies is a research problem that has attired the attention of the security research community in the recent years. Consequently, different approaches to tackle the problem have been proposed.

From a formal focus, in [31] the authors provide the foundations of a formal framework able to represent policies in different architectural layers and the dependencies between layers. Similarly, in [22] the authors analyze diverse combination needs for access control policies. Among them, the combination of heterogeneous policies and the integration of hierarchical policies through refinement. Algebraic operations for representing and manipulate these combinations are provided. In [82] Method-B is used to formalize the deployment of access control policies on systems composed by several (network) components. The authors also define and implement security properties to check the correctness the deployment process. Finally, by using model-driven techniques, in [39] the authors formalize what they call the policy continuum model, representing policies at different inter-related abstraction layers although it does not tackle the problem of inter-related architectural layers.

None of these formalization works provide the bridges necessary to fill the gap between real policies and the proposed formalisms and they mostly aim at providing a formal framework to deploy/analyse/manipulate synthetic policies. Conversely, our approach works the other way round by proposing a more pragmatic approach,

aimed at providing a solution for the integration of real, already deployed policies. Moreover, in order to ease its adoptability, we base our approach in the use of existing standards and off-the-shelf proved model-driven tools.

More similar to us, in [65] they describe some interesting results regarding the integration of policies not belonging to the same authorization entity. However, they do not deal with dependencies between policies, what is our focus here. They propose a similarity process to compare different rules and policies that we have adapted here for the case of inter-dependent access control policies.

Not focused on access-control, [30] provide an approach to detect conflicts between different kinds of policies in the same environment A.C policies, copyright policies, etc.

Conclusion

10.1 Conclusion

Building an information system often requires the composition of several collaborating subsystems. When security requirements are to be met, all the subsystems capable of participating in the security enforcement must be properly configured, not only to protect the information they manage, but also to collaborate with other subsystems. Failing to do so will put the system under the risk of unintended data disclosures, but also under the risk of not being able to function properly.

In the concrete case of data confidentiality and integrity, access-control mechanisms have been integrated in a vast diversity of components. This way, it is not uncommon to find a system where a number of subsystems implement and enforce access-control policies.

However, despite the few methods intended to automatically generate correct access-control policies from high level, verified representations, current access-control implementation in concrete components remains a complex task. Low level, often vendor-specific technologies used under the expertise and flair of security administrators fall short to assure the deployment of correct security policies. Moreover, the relations between the diverse subsystems are not explicitly taken into account when developing the subsystem policies, leading to unexpected behaviours.

In this scenario, a mechanism to discover the policies already deployed in a given information system turns up as a critical necessity. Unfortunately, while a vast amount of works have been devoted to the analysis of access-control policies, most of them worked on abstract, synthetic policies, disregarding the connection to the real deployed policies. In the same sense, reverse engineering methods for ex-

tracting information out of concrete systems have been proposed, however, security aspects have been traditionally ignored in those methods.

This thesis has presented a method for covering this gap. We have presented a model-driven reverse engineering approach aimed at extracting access-control policies from deployed information systems.

Our method is divided in two steps. First, it extracts the access-control policy deployed in concrete components, raising the level of abstraction so that low-level and vendor-specific details can be disregarded. Then, analysis tasks have been performed at this abstraction level, showing how model-driven techniques help to manipulate and analyse information in a standard and reusable way.

As a second step, our method gathers all the access-control policies being enforced in a given system and integrates them in a single model. We do so by unveiling the functional dependencies between the corresponding subsystems, showing how these dependencies also hold with respect to the access-control policies. Having all the policies integrated in a single model representation along with their dependencies enables the description and development of analysis tasks, unavailable when looking to the individual policies as isolated. Concretely, intra-component misconfigurations, i.e., combinations of rules in different policies leading to unexpected behaviours, can be unveiled.

Using model-driven techniques to raise the abstraction level of the deployed policies we assure the re-usability and uniformity of the analysis and applications we have proposed. Moreover, it allows us to see our contribution as complementary to other ones. Our approach can be joined, by model transformation and model composition, to other model-driven reverse engineering approaches, in order to complement them with security information. Then, the representations we provide can serve as a bridge to other representations, enabling the reusability of existing access-control analysis techniques.

10.2 Future Work

In this section, we would like to introduce four possible research lines we envisage as further research. The first three are intended to extend our work to deal with: 1) More components, lying in different architecture layers, 2) Other sources of information, like logs and audits and 3) different kinds of policies and not only access-control policies. The last one pretends to further test the scalability of the models and operations proposed in this thesis work by applying them to real, industrial-size scenarios.

Each of this research lines is sketched below.

10.2.1 Adding other components: The SeLinux case

In the present work, access-control policies have been extracted from three different components lying in three different architecture layers. As a future research line we intend to extend our work to extract access-control policies from other systems lying in another architecture layer different that the ones already tackled here. Specially interesting is to analyse the operating system layers, as many other system depend on these layers. Concretely, we intend to apply our model-driven method and techniques to the extraction of access-control policies from SeLinux systems.

Security-Enhanced Linux (SELinux) is a Linux kernel security module that provides the mechanism for supporting access control security policies, including mandatory access control (MAC)[90]. However, the complexity of the systems prevents users to take advantage of its powerful mechanisms. Among the rules composing a security policy configuration, many relationships occur and it is extremely difficult to understand their overall effects in the system [102].

Although some formal works try to provide the means to generate and analyze SeLinux security policies [102], [47] and even some approaches are intended to recover those policies from deployed systems [103] there is room for applying a model-driven reverse engineering approach as those solutions are only partial and moreover, they do not integrate the results into a global model as the one we have proposed here. By doing so, the SeLinux rules could be taken into account when analysing misconfigurations between components improving the overall picture of the multi-layer access-control analysis.

10.2.2 Other sources of informations: Logs and Audits

The access-control models we extract and the integration process we propose allows for the discovery of possible misconfigurations (both, inter and intra-component). In some cases, the misconfigurations detected will constitute warnings more than errors, as it will depend on the capacity of exploiting the error for it to be categorized

as such. In this sense, logs and audits are useful tools, as they register the real usage of the systems, disregarding the policy in place.

Analyzing such information sources, could help to 1) really categorize misconfigurations as being exploited configuration errors 2) detect discrepancies between the enforced policy and the real usage. Thus, it allows for misuse and intrusion detection analysis. Such analysis have been already proved useful for, among others, the domain of databases [53] and web-based applications[55].

We aim at applying the same method we have used to extract AC models to the extraction of data from audits and logs. By combining both models, the models from the policies and the models from the audits we believe useful information can be obtained. Analysis techniques as the ones mentioned above can be implemented over the models and what is more interesting, the results of those analysis can be related to the security policy, so that the conflicting parts can be more easily detected and modified.

Apart from misuse and intrusion detection, we also believe the analysis of the policy and the audits together will yield information that could lead to the refactoring of the policy in order to reduce its complexity and improve its efficiency. As an example, unused roles or privileges could be removed from the system policy or some other roles could be refactored to have less privileges.

10.2.3 Including different kinds of policies

The work of this thesis have been focused on access-control, secrecy, policies. However, other kind of policies exists within information systems. Privacy [58], [13] (what customer's data is stored by whom, for what purpose, for what duration, and with whom it is shared), and Integrity (is the data modified within a workflow by unauthorized parties?) policies among others.

We believe that our results can be directly applicable to the extraction of these different kind of policies.

With models representing the deployed policies available, we intent to explore the possible integration between the different policies. Similar to our analysis in Chapter 8, we believe that anomalies and misconfigurations could appear when several different kinds of policies work together in a given information system.

We propose the following roadmap:

1. Extract integrity privacy and availability policies from different components using them. This step may include the development of domain-specific models able to represent the information of a given policy within a given domain, abstracting from concrete implementation technologies (such domain-specific language already exist, like the P3P [36] standard for web sites).

2. Discovery of possible relations between policies. For that purpose, methods as the policy similarity calculation in [65] could be extended.
3. With the discovered dependencies, integration of the policies and global analysis for unveiling conflicts and misconfiguration between policies. Here it is interesting to remark that XACML, the language we proposed as integration language have been used for storing privacy policies [48], what could facilitate our integration goals.

10.2.4 Scalability of the approach

All along this thesis, new languages, visualizations, analysis and integration algorithms have been provided. In order to validate them, prototype tools have been developed and examples have been tested on them. Unfortunately, the examples we tested, while sometimes coming from the real world were partial or small.

The scalability (in computation time, but also in understandability) remain to be tested. To achieve this goal, the use of real, industrial-size examples arises as a clear necessity.

While obtaining such examples remains challenging, mostly due to the sensitivity of the problem in hand, as a next step we intend to contact enterprises interested in our results to test them in their systems (to that purpose, some consultancy relations have already been established). As a first step, we are already planning the testing of our network tools on the network configuration files (previously anonymized) of a big university. With this information available, a cycle of refining of the developed tools is to be launched.

List of Publications

International peer-reviewed journals:

- [1] Joaquín García-Alfaro, Frédéric Cuppens, Nora Cuppens-Boulahia, Salvador Martínez, Jordi Cabot: **Management of stateful firewall misconfiguration**. In *Computers and Security* 39. (2013) (64-85)

International peer-reviewed conferences:

- [2] Salvador Martínez, Valerio Cosentino, Jordi Cabot, Frédéric Cuppens: **Reverse Engineering of Database Security Policies**. In *Database and Expert Systems Applications - 24th International Conference, DEXA 2013, Prague, Czech Republic, August 26-29. (2013) (442-449)*
- [3] Salvador Martínez, Joaquín García-Alfaro, Frédéric Cuppens, Nora Cuppens-Boulahia, Jordi Cabot: **Model-Driven Extraction and Analysis of Network Security Policies**. In *Model-Driven Engineering Languages and Systems - 16th International Conference, MODELS 2013, Miami, FL, USA, September 29 - October 4. (2013) (52-68)*

International peer-reviewed workshops:

- [4] Salvador Martínez, Jordi Cabot, Joaquín García-Alfaro, Frédéric Cuppens, Nora Cuppens-Boulahia: **A model-driven approach for the extraction of network access-control policies**. In *Model-Driven Security Workshop, MDsec 2012, In conjunction with MoDELS 2012, Innsbruck, Austria, October 1. (2012)*
- [5] Salvador Martínez, Joaquín García-Alfaro, Frédéric Cuppens, Nora Cuppens-Boulahia, Jordi Cabot: **Towards an Access-Control Metamodel for Web Content Management Systems**. In *9th International Workshop on Model-Driven and Agile Engineering for the Web (MDWE) Aalborg, Denmark, July 8-12. (2013) (148-155)*

Not directly on the thesis subject, collaborations in the AtlanMod team have lead to several contributions in the field of model-driven engineering:

- [6] Massimo Tisi, Salvador Martínez, Hassene Choura: **Parallel Execution of ATL Transformation Rules**. In Model-Driven Engineering Languages and Systems - 16th International Conference, MODELS 2013, Miami, FL, USA, September 29 - October 4. (2013) (656-672)

- [7] Valerio Cosentino, Salvador Martínez: **Extracting UML/OCL Integrity Constraints and Derived Types from Relational Databases**. In 13th International Workshop on OCL, Model Constraint and Query Languages, MODELS 2013, Miami, FL, USA, September 29 - October 4. (2013) (43-52)

- [8] Manuel Wimmer, Salvador Martínez, Frédéric Jouault, Jordi Cabot: **A Catalogue of Refactorings for Model-to-Model Transformations**. In Journal of Object Technology. (2012) (1-40)

Bibliography

- [1] Nmap 6. The Network Mapper. 67
- [2] QueSO. Remote O.S. detector. 67
- [3] The NetFilter Project: Firewalling, NAT and Packet Mangling for linux. 67, 68
- [4] Trafscrambler. Defeating sniffers and Intrusion Detection Systems using forged TCP resets. 67
- [5] *Building secure software: how to avoid security problems the right way*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002. 51
- [6] Drupal Open-source CMS. <http://drupal.org/>, 2013. 96
- [7] Firewall Reverse Engineering project web site. http://www.emn.fr/z-info/atlanmod/index.php/Firewall_Reverse_Engineering, 2013. 54, 73
- [8] D. 5200.28-STD. *Trusted Computer System Evaluation Criteria*. Dod Computer Security Center, December 1985. 24, 31
- [9] A. Abou El Kalam, R. E. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. In *4th IEEE International Workshop on Policies for Distributed Systems and Networks (Policy'03)*, pages 120 – 131, June 2003. 25, 31
- [10] M. H. Alalfi, J. R. Cordy, and T. R. Dean. Recovering role-based access control security models from dynamic web applications. In *Web Engineering*, pages 121–136. Springer, 2012. 36, 95, 101, 134
- [11] M. Alanen and I. Porres. Difference and union of models. In *UML*, volume 2863 of *LNCS*, pages 2–17. Springer, 2003. 91
- [12] M. Arlitt and C. Williamson. An analysis of TCP reset behaviour on the internet. *Computer Communication Review*, 35(1):37–44, 2005. 66
- [13] P. Ashley, S. Hada, G. Karjoth, and M. Schunter. E-p3p privacy policies and privacy authorization. In *Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*, pages 103–109. ACM, 2002. 140

- [14] I. Astrova. Towards the semantic web - an approach to reverse engineering of relational databases to ontologies. In *ADBIS Research Communications*, volume 152 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005. [35](#), [134](#)
- [15] S. Barker and P. Douglas. RBAC policy implementation for SQL databases. In *DBSec*, pages 288–301, 2003. [35](#), [77](#)
- [16] Y. Bartal, A. Mayer, K. Nissim, and A. Wool. Firmato: A novel firewall management toolkit. *ACM Trans. Comput. Syst.*, 22(4):381–420, Nov. 2004. [33](#), [49](#), [65](#)
- [17] D. Basin, J. Doser, and T. Lodderstedt. Model driven security: From UML models to access control infrastructures. *ACM Trans. Softw. Eng. Methodol.*, 15:39–91, January 2006. [32](#), [77](#), [92](#)
- [18] N. Benaïssa, D. Cansell, and D. Méry. Integration of security policy into system modeling. In *Proceedings of the 7th international conference on Formal Specification and Development in B*, B’07, pages 232–247, Berlin, Heidelberg, 2006. Springer-Verlag. [32](#)
- [19] M. Benantar. *Access control systems: security, identity management and trust models*. Springer, 2006. [22](#)
- [20] E. Bertino and R. Sandhu. Database security-concepts, approaches, and challenges. *IEEE Trans. Dependable Secur. Comput.*, 2:2–19, January 2005. [35](#), [78](#)
- [21] M. Bishop and S. Peisert. Your security policy is what?? Technical report, 2006. [34](#), [130](#)
- [22] P. Bonatti, S. De Capitani di Vimercati, and P. Samarati. An algebra for composing access control policies. *ACM Transactions on Information and System Security (TISSEC)*, 5(1):1–35, 2002. [36](#), [135](#)
- [23] M. Brambilla, J. Cabot, and M. Wimmer. Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, 1(1):1–182, 2012. [25](#)
- [24] A. D. Brucker, L. Brügger, P. Kearney, and B. Wolff. Verified firewall policy transformations for test-case generation. In *Third International Conference on Software Testing, Verification, and Validation (ICST)*, pages 345–354. IEEE Computer Society, Los Alamitos, CA, USA, 2010. [34](#), [132](#)
- [25] A. D. Brucker and B. Wolff. Test-sequence generation with hol-testgen with an application to firewall testing. In *Tests and Proofs*, pages 149–168. Springer, 2007. [132](#)
- [26] H. Brunelière, J. Cabot, G. Dupé, F. Madiot, et al. Modisco: a model driven reverse engineering framework. *Information and Software Technology*, 2014. [33](#), [90](#)

- [27] L. Buttyan, G. Pék, and T. Thong. Consistency verification of stateful firewalls is not harder than the stateless case. *Infocommunications Journal*, LXIV(1):2–8, 2009. [34](#)
- [28] G. Canfora Harman and M. Di Penta. New frontiers of reverse engineering. FOSE '07, pages 326–341. IEEE Computer Society, 2007. [32](#), [78](#)
- [29] J. Canovas. *Lenguajes Específicos del Dominio para la extracción de modelos desde los Espacios Tecnológicos del grammarware, dataware y apiware*. PhD thesis, Universidad de Murcia, 2011. [28](#)
- [30] M. M. Casalino, H. Plate, and S. Trabelsi. Transversal policy conflict detection. In *Engineering Secure Software and Systems*, pages 30–37. Springer, 2012. [36](#), [136](#)
- [31] M. M. Casalino and R. Thion. Refactoring multi-layered access control policies through (de)composition. In *CNSM*, pages 243–250, 2013. [36](#), [135](#)
- [32] S. Ceri, P. Fraternali, and A. Bongio. Web modeling language (webml): a modeling language for designing web sites. *Computer Networks*, 33(1):137–157, 2000. [36](#)
- [33] R. H. L. Chiang, T. M. Barron, and V. C. Storey. Reverse engineering of relational databases: extraction of an EER model from a relational database. *Data Knowl. Eng.*, 12:107–142, March 1994. [35](#), [78](#), [134](#)
- [34] E. J. Chikofsky, J. H. Cross, et al. Reverse engineering and design recovery: A taxonomy. *Software, IEEE*, 7(1):13–17, 1990. [29](#), [32](#)
- [35] V. Cosentino. *A model-based approach for extracting business rules out of legacy information systems*. PhD thesis, Ecole des Mines de Nantes, 2013. [26](#), [27](#)
- [36] L. Cranor, M. Langheinrich, M. Marchiori, and J. Reagle. The platform for privacy preferences 1.0 (p3p1.0) specification. W3C Recommendation, Apr. 2002. [140](#)
- [37] F. Cuppens, N. Cuppens-Boulahia, J. Garcia-Alfaro, T. Moataz, and X. Riomasson. Handling Stateful Firewall Misconfiguration. In *27th IFIP International Information Security and Privacy Conference (SEC 2012)*, pages 174–186, 2012. [34](#), [70](#)
- [38] F. Cuppens, N. Cuppens-Boulahia, T. Sans, and A. Miège. A formal approach to specify and deploy a network security policy. In *Formal Aspects in Security and Trust'04*, pages 203–218, 2004. [33](#), [115](#)
- [39] S. Davy, B. Jennings, and J. Strassner. The policy continuum—policy authoring and conflict analysis. *Computer Communications*, 31(13):2981–2995, 2008. [36](#), [135](#)

- [40] J. Garcia-Alfaro, N. Boulahia-Cuppens, and F. Cuppens. Complete analysis of configuration rules to guarantee reliable network security policies. *Int. J. Inf. Secur.*, 7(2):103–122, Mar. 2008. [34](#), [59](#), [62](#)
- [41] J. Garcia-Alfaro, F. Cuppens, and N. Cuppens-Boulahia. Aggregating and deploying network access control policies. volume 0, pages 532–542, Los Alamitos, CA, USA, 2007. IEEE Computer Society. [62](#)
- [42] J. Garcia-Alfaro, F. Cuppens, and N. Cuppens-Boulahia. Management of exceptions on access control policies. In *SEC*, volume 232 of *IFIP*, pages 97–108. Springer, 2007. [56](#)
- [43] J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia, S. Martinez, and J. Cabot. Management of stateful firewall misconfiguration. *Computers & Security*, (0):–, 2013. [37](#)
- [44] F. Gauthier, D. Letarte, T. Lavoie, and E. Merlo. Extraction and comprehension of moodle’s access control model: A case study. In *(PST), 2011*, pages 44–51. IEEE, 2011. [36](#), [95](#), [101](#), [134](#)
- [45] S. Goel and I. N. Chengalur-Smith. Metrics for characterizing the form of security policies. *The Journal of Strategic Information Systems*, 19(4):281 – 295, 2010. [91](#)
- [46] M. Gouda and A. Liu. A model of stateful firewalls and its properties. In *35th International Conference on Dependable Systems and Networks (DSN 2005)*, pages 128–137, 2005. [34](#)
- [47] B. Hicks, S. Rueda, L. St Clair, T. Jaeger, and P. McDaniel. A logical specification and analysis for selinux mls policy. *ACM Transactions on Information and System Security (TISSEC)*, 13(3):26, 2010. [139](#)
- [48] W. Hommel. Using xacml for privacy control in saml-based identity federations. In *Communications and Multimedia Security*, pages 160–169. Springer, 2005. [141](#)
- [49] O. Is. ISO/IEC FCD 9075-14 (SQL/XML), 2007. [79](#)
- [50] X. Jin, R. Krishnan, and R. Sandhu. A unified attribute-based access control model covering dac, mac and rbac. In *Data and Applications Security and Privacy XXVI*, pages 41–55. Springer, 2012. [31](#), [112](#)
- [51] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. ATL: A model transformation tool. *Science of Computer Programming*, 72(1):31–39, 2008. [12](#), [29](#), [44](#), [74](#)
- [52] J. Jürjens. UMLsec: Extending UML for secure systems development. UML ’02, pages 412–425. Springer, 2002. [32](#), [132](#)
- [53] A. Kamra, E. Terzi, and E. Bertino. Detecting anomalous access patterns in relational databases. *The VLDB Journal*, 17(5):1063–1077, 2008. [140](#)

- [54] D. Kolovos, D. Di Ruscio, A. Pierantonio, and R. Paige. Different models for model matching: An analysis of approaches to support model differencing. In *Comparison and Versioning of Software Models, 2009. CVSM '09. ICSE Workshop on*, pages 1–6, 2009. 91
- [55] C. Kruegel and G. Vigna. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 251–261. ACM, 2003. 140
- [56] M. Kuhlmann, K. Sohr, and M. Gogolla. Comprehensive two-level analysis of static and dynamic rbac constraints with uml and ocl. In *Secure Software Integration and Reliability Improvement (SSIRI), 2011 Fifth International Conference on*, pages 108–117, june 2011. 32
- [57] I. Kurtev, J. Bézivin, and M. Aksit. Technological spaces: An initial appraisal. In *CoopIS, DOA'2002 Federated Conferences, Industrial track*, 2002. 28, 84
- [58] R. Lämmel and E. Pek. Understanding privacy policies - a study in empirical analysis of language usage. *Empirical Software Engineering*, 18(2):310–374, 2013. 140
- [59] H. Lockhart, B. Parducci, and A. Anderson. OASIS XACML TC, 2013. 14, 46, 108, 112
- [60] H. Lockhart, B. Parducci, and E. Rissanen. XACML v3.0 Core and Hierarchical Role Based Access Control (RBAC) Profile Version 1.0, 2010. 113
- [61] T. Lodderstedt, D. A. Basin, and J. Doser. SecureUML: A UML-based modeling language for model-driven security. In *Proceedings of the 5th International Conference on The Unified Modeling Language, UML '02*, pages 426–441. Springer, 2002. 32, 112, 132
- [62] J. luc Hainaut, M. Chandelon, M. Ch, C. Tonneau, M. Joris, J. l. Hainaut, M. Ch, C. Tonneau, and M. Joris. Contribution to a theory of database reverse engineering. In *in Proc. of the IEEE Working Conf. on Reverse Engineering*, pages 161–170. IEEE Computer Society, 1993. 35, 78, 134
- [63] S. Martínez, J. Cabot, J. Garcia-Alfaro, F. Cuppens, and N. Cuppens-Boulahia. A model-driven approach for the extraction of network access-control policies. In *Proceedings of the Workshop on Model-Driven Security, MDsec '12*, pages 5:1–5:6. ACM, 2012. 37, 54
- [64] A. Mayer, A. Wool, and E. Ziskind. Fang: A firewall analysis engine. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy, SP '00*, pages 177–, Washington, DC, USA, 2000. IEEE Computer Society. 34, 128, 129, 130

- [65] P. Mazzoleni, B. Crispo, S. Sivasubramanian, and E. Bertino. Xacml policy integration algorithms. *ACM Transactions on Information and System Security (TISSEC)*, 11(1):4, 2008. [36](#), [120](#), [136](#), [141](#)
- [66] M. Meike, J. Sametinger, and A. Wiesauer. Security in open source web content management systems. *Security & Privacy, IEEE*, 7(4):44–51, 2009. [36](#), [95](#), [134](#)
- [67] X. Meng, G. Jiang, H. Zhang, H. Chen, and K. Yoshihira. Automatic Profiling of Network Event Sequences: Algorithm and Applications. In *27th Conference on Computer Communications (INFOCOM 2008)*, pages 1–9, 2008. [66](#)
- [68] T. Mouelhi, F. Fleurey, B. Baudry, and Y. L. Traon. A model-based framework for security policy specification, deployment and testing. In *MoDELS 2008*, pages 537–552, 2008. [112](#)
- [69] T. Nelson, C. Barratt, D. J. Dougherty, K. Fisler, and S. Krishnamurthi. The margrave tool for firewall analysis. In *Proceedings of the 24th international conference on Large installation system administration, LISA'10*, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association. [34](#), [129](#), [132](#)
- [70] S. Oh and S. Park. Enterprise model as a basis of administration on role-based access control. In *CODAS'01*, pages 165–174, 2001. [35](#), [77](#), [92](#)
- [71] OMG. *Object Constraint Language Specification, version 2.0*. Object Management Group, June 2005. [13](#), [45](#), [90](#), [108](#)
- [72] OMG. *Meta Object Facility (MOF) Core Specification Version 2.0*, 2006. [27](#)
- [73] OMG. *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Version 1.1*, January 2011. [29](#)
- [74] OMG. *OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1*, August 2011. [27](#)
- [75] Oracle. *Oracle database security guide 11g release 1 (11.1)*, 2011. [81](#)
- [76] S. M. Perez, V. Cosentino, J. Cabot, and F. Cuppens. Reverse engineering of database security policies. In *DEXA (volume 2)*, pages 442–449, 2013. [37](#), [100](#)
- [77] S. M. Perez, J. García-Alfaro, F. Cuppens, N. Cuppens-Boulahia, and J. Cabot. Model-driven extraction and analysis of network security policies. In *MoDELS*, pages 52–68, 2013. [37](#), [115](#)
- [78] S. M. Perez, J. García-Alfaro, F. Cuppens, N. Cuppens-Boulahia, and J. Cabot. Towards an access-control metamodel for web content management systems. In *ICWE Workshops*, pages 148–155, 2013. [37](#)

- [79] J.-M. Petit, J. Kouloumdjian, J.-F. Boulicaut, and F. Toumani. Using queries to improve database reverse engineering. *ER '94*, pages 369–386. Springer, 1994. [35](#), [134](#)
- [80] S. Pozo, R. Ceballos, and R. M. Gasca. Model-based development of firewall rule sets: Diagnosing model inconsistencies. *Inf. Softw. Technol.*, 51(5):894–915, May 2009. [33](#), [131](#)
- [81] S. Pozo, R. Gasca, A. Reina-Quintero, and A. Varela-Vaca. Confident: A model-driven consistent and non-redundant layer-3 firewall acl design, development and maintenance framework. *Journal of Systems and Software*, 85(2):425 – 457, 2012. [33](#), [49](#), [65](#), [131](#), [132](#)
- [82] S. Preda, N. Cuppens-Boulahia, F. Cuppens, J. García-Alfaro, and L. Toutain. Model-driven security policy deployment: Property oriented approach. In *ESSoS*, pages 123–139, 2010. [32](#), [36](#), [135](#)
- [83] S. Rugaber and K. Stirewalt. Model-driven reverse engineering. *Software, IEEE*, 21(4):45 – 53, july-aug. 2004. [33](#)
- [84] R. Russell. Linux 2.4 packet filtering howto. <http://www.netfilter.org/documentation/HOWTO/packet-filtering-HOWTO.html>, 2002. [51](#)
- [85] 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, New York, NY, USA, 2000. ACM. [24](#), [31](#), [77](#)
- [86] R. S. Sandhu and P. Samarati. Access control: principle and practice. *Communications Magazine, IEEE*, 32(9):40–48, 1994. [22](#)
- [87] R. C. seacord, D. Plakosh, and G. A. Lewis. *Modernizing Legacy Systems: Software Technologies, Engineering Process and Business Practices*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003. [32](#)
- [88] E. A. Shaer and H. Hamed. Modeling and management of firewall policies. *IEEE Trans. Network and Service Management*, pages 2 – 10, Apr. 2004. [34](#)
- [89] M. Shehab, S. Al-Haj, S. Bhagurkar, and E. Al-Shaer. Anomaly discovery and resolution in MySQL access control policies. volume 7447 of *LNCS*, pages 514–522. Springer, 2012. [35](#), [133](#), [134](#)
- [90] S. Smalley, C. Vance, and W. Salamon. Implementing selinux as a linux security module. *NAI Labs Report*, 1:43, 2001. [139](#)
- [91] E. Soler, R. Villarroel, J. Trujillo, E. Fernandez-Medina, and M. Piattini. Representing security and audit rules for data warehouses at the logical level

- by using the common warehouse metamodel. In *Proceedings of the First International Conference on Availability, Reliability and Security*, pages 914–921. IEEE Computer Society, 2006. [35](#), [92](#)
- [92] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley, Boston, MA, 2. edition, 2009. [93](#), [123](#)
- [93] M. Tisi, S. Martínez, F. Jouault, and J. Cabot. Refining Models with Rule-based Model Transformations. Rapport de recherche RR-7582, INRIA, 2011. [74](#)
- [94] A. Tongaonkar, N. Inamdar, and R. Sekar. Inferring higher level policies from firewall rules. In *Proceedings of the 21st conference on Large Installation System Administration Conference, LISA'07*, pages 2:1–2:10, Berkeley, CA, USA, 2007. USENIX Association. [34](#), [127](#), [128](#)
- [95] J. Treurniet. Detecting Lowprofile Scans in TCP Anomaly Event Data. In *Fourth Annual Conference on Privacy, Security and Trust (PST 2006)*, pages 1–8, 2006. [67](#)
- [96] B. Trninic, G. Sladic, G. Milosavljevic, B. Milosavljevic, and Z. Konjovic. Policydsl: Towards generic access control management based on a policy metamodel. In *SoMeT*, pages 217–223, 2013. [112](#)
- [97] G. Vaidyanathan and S. Mautone. Security in dynamic web content management systems applications. *Communications of the ACM*, 52(12):121–125, 2009. [36](#), [95](#), [134](#)
- [98] E. Visser, J. Warmer, A. Van Deursen, and A. Van Deursen. Model-driven software evolution: A research agenda. In *In Proc. Int. Ws on Model-Driven Software Evolution held with the ECSMR'07*, 2007. [33](#)
- [99] E. Yuan and J. Tong. Attributed based access control (abac) for web services. In *Proceedings of the IEEE International Conference on Web Services, ICWS '05*, pages 561–569, Washington, DC, USA, 2005. IEEE Computer Society. [25](#), [31](#), [112](#)
- [100] L. Yuan and H. Chen. Fireman: a toolkit for firewall modeling and analysis. In *In Proceedings of IEEE Symposium on Security and Privacy*, pages 199–213, 2006. [33](#)
- [101] V. Zaliva. Platform-independent firewall policy representation. *CoRR*, abs/0805.1886, 2008. [33](#), [130](#)
- [102] G. Zanin and L. V. Mancini. Towards a formal model for security policies specification and validation in the selinux system. In *Proceedings of the Ninth ACM Symposium on Access Control Models and Technologies, SACMAT '04*, pages 136–145, New York, NY, USA, 2004. ACM. [139](#)

- [103] G. Zhai, W. Ma, M. Tian, N. Yang, C. Liu, and H. Yang. Design and implementation of a tool for analyzing selinux secure policy. In *Proceedings of the 2Nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, ICIS '09, pages 446–451, New York, NY, USA, 2009. ACM. [139](#)

Thèse de Doctorat

Salvador MARTÍNEZ

Analyse et Reconstruction Automatique de Politiques de Sécurité de Composants de Sécurité Déployés

Automatic Reconstruction and Analysis of Security Policies from Deployed Security Components

Résumé

La sécurité est une préoccupation essentielle pour tout système d'information. Propriétés de sécurité telles que la confidentialité, l'intégrité et la disponibilité doivent être appliquées afin de rendre les systèmes sûrs. Dans les environnements complexes, où les systèmes d'information sont composés par un certain nombre de sous-systèmes hétérogènes, chaque sous-système joue un rôle clé dans la sécurité globale du système. Dans le cas spécifique du contrôle d'accès, politiques de contrôle d'accès peuvent être trouvées dans différents composants (bases de données, réseaux, etc.), ces derniers étant sensés travailler ensemble. Néanmoins, puisque la plupart de ces politiques ont été mises en œuvre manuellement et / ou évolué séparément ils deviennent facilement incompatibles. Dans ce contexte, la découverte et compréhension des politiques de sécurité appliquées par le système d'information devient une nécessité critique. Le principal défi à résoudre est de combler le fossé entre les caractéristiques de sécurité dépendant du fournisseur et une représentation de plus haut niveau que exprime ces politiques d'une manière faisant abstraction des spécificités de composants concrets, et donc, plus facile à comprendre et à raisonner avec. Cette représentation de haut niveau nous permettrait également de mettre en œuvre tous les opérations de évolution / refactoring / manipulation sur les politiques de sécurité d'une manière réutilisable. Dans ce travail, nous proposons un tel mécanisme de rétro-ingénierie et d'intégration des politiques de contrôle d'accès. Nous comptons sur les technologies de l'ingénierie dirigée par les modèles pour atteindre cet objectif.

Mots clés

Sûreté, Ingénierie dirigée par les modèles, Rétro-ingénierie, Contrôle d'accès.

Abstract

Security is a critical concern for any information system. Security properties such as confidentiality, integrity and availability need to be enforced in order to make systems safe. In complex environments, where information systems are composed by a number of heterogeneous subsystems, each subsystem plays a key role in the global system security. For the specific case of access-control, access-control policies may be found in several components (databases, networks and applications) all, supposedly, working together. Nevertheless since most times these policies have been manually implemented and/or evolved separately they easily become inconsistent. In this context, discovering and understanding which security policies are actually being enforced by the information system comes out as a critical necessity. The main challenge to solve is bridging the gap between the vendor-dependent security features and a higher-level representation that express these policies in a way that abstracts from the specificities of concrete system components, and thus, it's easier to understand and reason with. This high-level representation would also allow us to implement all evolution/refactoring/manipulation operations on the security policies in a reusable way. In this work we propose such a reverse engineering and integration mechanism for access-control policies. We rely on model-driven technologies to achieve this goal.

Key Words

Security, Model-driven, Reverse-engineering, Access-Control.