Refactoring Multi-Layered Access Control Policies Through (De)Composition

Matteo Maria Casalino*†
*SAP Labs France
Sophia-Antipolis, France
matteo.maria.casalino@sap.com

Romuald Thion†
†Université Claude Bernard Lyon 1
LIRIS CNRS UMR5205, France
romuald.thion@liris.cnrs.fr

***Abstract*—Policy-based access control is a well-established paradigm for securing layered IT systems. Access control policies, however, often do not focus on dedicated architecture layers, but increasingly employ concepts of multiple layers. Web application servers, for instance, typically support request filtering on the basis of network addresses. The resulting flexibility comes with increased management complexity and the risk of security-relevant misconfiguration when looking at the various policies in isolation. We therefore propose a flexible access control framework able to provide a comprehensive view of the global access control policy implemented in a given system. The focus of this paper is to lay down the theoretical foundations of this framework that allows (i) to describe authorization policies from different architecture layers, (ii) to capture the semantics of dependencies between layers in order to create a composed view of the global policy, and (iii) to decompose the global policy again into a collection of simpler ones by means of algebraic techniques inspired from database normalization theory.**

I. Introduction

Access control is pervasive within the different layers of IT infrastructures. For instance, network filtering and application-layer authorization policies are different forms of access control that typically cooperate in real scenarios. The access control process is distributed across several IT components, each one potentially operating on different architecture layers and residing on different hosts. For instance, a classical network firewall is able to take allow/deny decisions for network requests, having parameters such as IP addresses and TCP/UDP ports. A Web server instead handles a different kind of requests that rather belong to the application ISO/OSI layer, e.g., having parameters such as the requested URL.

However, the separation between network and application layers is typically not as neat. For example, many common services (e.g., the Apache Web server, the MySQL database server or the anti-spam features of the sendmail mail server), perform access control based not only on application-specific parameters (e.g., respectively, URLs, tables, mail addresses), but can overlap with the lower layer (e.g., by filtering on the IP address of the requester). Conversely, modern firewalls are more and more capable of inspecting application layer fields.

Such inter-layer relationships can increase system security, in practice, however, more complexity often leads to misconfiguration, compromising systems security [1]. Accordingly, the problem of distributed inter-dependent policies is identified as one of the security automation research challenges in [2].

In this paper we focus on *inter-layer policy refactoring*, i.e., the task of finding the least permissive rewriting of a collection of policies that belong to different layers such that the global composed policy remains identical. Policy refactoring is a means (i) to enforce the least privilege principle in multi-layered policy-based access control systems, (ii) to reduce management overhead by simplifying local policies and (iii) to adapt to changing security capabilities of single components.

A. Motivating Scenario

In order to illustrate refactoring, we consider the following hypothetical scenario. Suppose a Web server featuring the capability to filter on the clients' IP addresses that is harbored in a network whose access from the Internet is mediated by a firewall. As the policies of the two devices are written independently, some unnecessary privileges may be granted by either of them. For instance, the firewall policy may be granting access to a larger portion of clients than actually allowed by the Web server policy or vice versa. Refactoring exploits the knowledge of the inter-layer interactions to reduce such privileges to the minimum, by preserving the composite policy. As such, unauthorized access attempts are blocked as soon as possible, according to the least privilege principle.

Suppose now that we are interested in replacing the Web server policy by a simpler one that does not discriminate clients' IP addresses while keeping the global composite policy unaltered. We would need to transfer part of the Web server policy to the lower layer, such that all the network-layer filtering is then performed by the firewall only. Whether such a decomposition is possible at all, depends both on the internal structure of the original policies and on the access control capabilities of the devices. For instance, if the Web server policy prevents a client from accessing only some specific URLs, the firewall cannot enforce it on its own unless it is capable of HTTP header inspection. In this case refactoring consists in first determining if the new desired access control layers layout can enforce the global policy and then finding how the original policies are to be rewritten.

B. Contribution

Our contributions are as follows. In Section II we define a model that captures the access control behavior of a collection of policy decision points that cooperate on different

ISBN 978-3-901882-53-1, 9th CNSM and Workshops ©2013 IFIP

TABLE I
EXAMPLE OF FIELDS AND RELATED DOMAINS

f	Meaning	$\mathrm{dom}(f)$
$\mathsf{I_s}$	IP source address	Integers range $[0, 2^{32}-1]$
$\mathsf{I_d}$	IP destination address	Integers range $[0, 2^{32}-1]$
$\mathsf{P_s}$	Port source	Integers range $[0, 2^{16}-1]$
$\mathsf{P_d}$	Port destination	Integers range $[0, 2^{16}-1]$
H	HTTP Host Header	Dot-separated strings
U	URL of HTTP Requests	Strings complying with rfc1738
$\sharp$	Singleton field	$\{\Box\}$

layers of the same IT infrastructure. In Section III we define composition of inter-dependent policies as the operation that, given a pair of access control decision functions, produces a composite decision function, decomposition being its inverse. In Section IV we provide a compact representation for our model based on which we devise algorithms that compute policy (de)composition and that we formally prove correct. In Section V we identify a criterion inspired from database normalization theory which characterizes precisely when policies can be decomposed and we show how it can be computed on our model. Moreover, we show how the proposed framework can be employed to solve the refactoring problem.

II. ACCESS CONTROL LAYERS

In this section we lay the foundations of a model that captures the access control policy implemented by the collection of policy decision points that operate at different layers within the same IT infrastructure (e.g., firewalls, application servers, Web servers, database servers, etc.). In particular, we aim at characterizing each such layer in terms of its access control capabilities and its interface with the other layers.

We rely on a classic and general description of access control systems, where a logical subsystem (usually called *policy decision point*) associates, for a given policy, a unique *decision* to any possible *access control request* [4], [5].

Once we come to reason about the composition of layers, we need to consider relations between the different types of requests they handle. For instance, the IP and port destination fields of the requests handled by a firewall are related to the IP addresses and ports of available services (e.g., Web and application servers). Intuitively, a particular firewall, depending on its policy, either enables requests to be further processed by other policy decision points or block them right away. Keeping track of these relationships allows to determine how decisions taken by one layer's policy decision points influence the ones taken in other layers. In order to formalize the above concepts we start from access control request fields and types.

Definition 1 (Request Field and Field's Domain): The finite set $\mathcal{F}$ is the universe of all request fields. Each field $f \in \mathcal{F}$ has a corresponding domain, written $\mathrm{dom}(f)$, that is the set of all possible values f can take in a request.

Table I presents some example request fields which we will refer to throughout the paper, together with their respective domains. The purpose of the special field $\sharp$ is to represent a fictitious singleton domain. A given set of fields identifies a type of access control requests, as stated in the next definition.

Definition 2 (Request Type and Request Space): A request type is a finite subset of request fields $F \in 2^{\mathcal{F}}$.

The request space $\mathfrak{Q}(F)$ associated with a request type $F = \{f_i\}_{i=1}^n$ characterizes all the requests existing over F. It is the Cartesian product of the domains of the fields in F: $\mathfrak{Q}(F) = \mathrm{dom}(f_1) \times \ldots \times \mathrm{dom}(f_n)$.

To remove ambiguity, we assume fields to be totally ordered and we say that the product of Definition 2 shall be taken according to this order. The same applies to any formula involving products of (functions of) fields in this paper.

Example request types are $F_{fw} = \{\mathsf{I_s}, \mathsf{I_d}, \mathsf{P_s}, \mathsf{P_d}, \mathsf{T_p}\}$, which characterizes requests handled by firewalls, or $F_{ws} = \{\mathsf{I_s}, \mathsf{H}, \mathsf{U}\}$ for application-layer requests for a Web server capable of filtering on the clients' IP address. Other combinations are possible too: for instance $F_{fw} \cup \{\mathsf{H}, \mathsf{U}\}$ models the request type of a firewall that can inspect parts of the HTTP header.

The definition of access control request follows directly from those of request type and request space. We define two operations on requests: concatenation and projection.

Definition 3 (Access Control Request): An access control request of type $F = \{f_i\}_{i=1}^n$ is an element of the request space $\mathfrak{Q}(F)$. The requests belonging to $\mathfrak{Q}(F)$ are therefore all the possible sequences $\langle v_1, \ldots, v_n \rangle$ with $v_i \in \mathrm{dom}(f_i)$. The ith coordinate of $q \in \mathfrak{Q}(F)$ is written $q(i) = v_i$.

Given the requests q_1 and q_2 having disjoint request types F_1, F_2, their concatenation is the request $q_1 + q_2$ (also denoted $q_1 q_2$) such that, $\forall f \in F_1 \cup F_2$, $(q_1 + q_2)(f) = q_i(f)$ if $f \in F_i$ (with $i \in \{1, 2\}$).

Given a request q, its projection on some subset P of its request type is denoted by $q|_P$ and it is the restriction of the sequence q to the fields in $P = \{p_1 \ldots p_n\}$: it is defined by $q|_P = \langle q(p_1) \ldots q(p_n) \rangle$.

We are now ready to provide a formal description of access control layers. An access control layer represents a collection of policy decision points that are all capable of processing access control requests of the same type. It conveys essentially the following three pieces of information:

- the type of access control requests that are in the layer's scope, i.e., those which the layer's decision points are deputed to express a decision for;
- how the request type handled locally relates to that of requests handled within other layers;
- which decision is taken, for every request, by any decision point in the layer according to its policy.

Definition 4 (Access Control Layer): An Access Control Layer (ACL) is a triple $\langle F, C, \delta \rangle$ where:

(i) *$F \in 2^{\mathcal{F}}$ is the layer request type;*
(ii) *$C \in 2^{\mathcal{F}} \setminus \{\emptyset\}$, s.t. $C \cap F = \emptyset$, is the layer coupling type;*
(iii) *$\delta : \mathfrak{Q}(C) \to (\mathfrak{Q}(F) \to D)$ is the access decision function with D the set of decisions.*

The coupling type C characterizes the interface with the layers an ACL can be composed with. Every value in the coupling space $c \in \mathfrak{Q}(C)$ identifies a policy decision point

TABLE II
EXAMPLE DECISION FUNCTION δ_a

$\mathsf{I_s}$	$\mathsf{I_d}$	$\mathsf{P_s}$	$\mathsf{P_d}$	D
1.1.1.∗	1.1.1.∗	∗	∗	1
2.2.∗.∗	1.1.1.1	∗	80	1
3.3.∗.∗	1.1.1.∗	∗	80	1
	...			0

TABLE III
EXAMPLE DECISION FUNCTION δ_b

$\mathsf{I_d}$	$\mathsf{P_d}$	$\mathsf{I_s}$	H	U	D
1.1.1.1	80	∗.∗.∗.∗ \ 3.3.3.∗	acme.com	/public/∗	1
		1.1.1.∗	acme.com	/private/∗	1
		2.2.2.∗	acme.com	/private/∗	1
		1.1.1.∗	beta.com	/∗	1
		2.2.3.∗	beta.com	/∗	1
			...		0
		...			⊥

in this layer; the function $\delta(c) : \mathfrak{Q}(F) \to D$, mapping every request $q \in \mathfrak{Q}(F)$ to a unique decision, represents its policy.

Suppose a scenario where a firewall FW protects the access to the IP network 1.1.1.0/24 in which resides a Web server WS listening on the address 1.1.1.1 and TCP port 80. Other services may possibly be running in the same IP network.

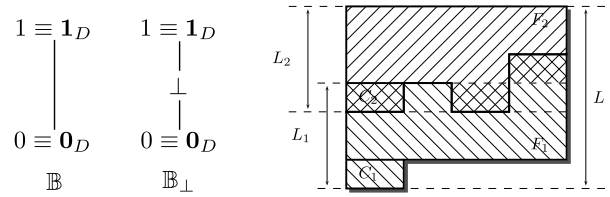
Depending on the access control capabilities of the policy decision points we consider, several configurations of access control layers are possible. Assume that WS supports IP address filtering. We then have $L_a = \langle \{\mathsf{I_s}, \mathsf{I_d}, \mathsf{P_s}, \mathsf{P_d}\}, \{\sharp\}, \delta_a \rangle$ and $L_b = \langle \{\mathsf{I_s}, \mathsf{H}, \mathsf{U}\}, \{\mathsf{I_d}, \mathsf{P_d}\}, \delta_b \rangle$, where a possible instance of δ_a (resp. δ_b) is reported in Table II (resp. Table III).

Each row in the tables maps all the requests matching to the wildcards to the decision reported in the last column. The set difference symbol "\" represents exceptions (e.g., ∗.∗.∗.∗ \ 3.3.3.∗ means every IP address except the subnet 3.3.3.0/24). The ellipsis "..." represents all the requests that do not match any other row. Note that the graph of decision functions is in principle huge or even infinite (depending on the fields' domains); we will deal with this issue in Section IV.

The policy decision points within L_b are uniquely identified by pairs in the coupling space $\mathfrak{Q}(\{\mathsf{I_d}, \mathsf{P_d}\}) = \mathrm{dom}(\mathsf{I_d}) \times \mathrm{dom}(\mathsf{P_d})$. As only one such decision point is known, namely the Web server WS identified by the pair $c_{ws} = \langle 1.1.1.1, 80 \rangle$, $\delta_b(c_{ws})$ is a completely definite function that represents the policy of WS. In contrast, $\delta_b(c) = \mathfrak{Q}(\{\mathsf{I_s}, \mathsf{H}, \mathsf{U}\}) \mapsto \bot$ for all $c \neq c_{ws}$, meaning that every other (unknown) decision point yields an undefined ($\bot$) decision for every possible request.

Not fixing a particular set of decisions in Definition 4 gives us flexibility to model different aspects of reality. In our running example, we illustrate this by making undefined behavior explicit, which leads to a form of partial knowledge reasoning. We argue that this eases the applicability of our approach to real world scenarios, where, even if it is not always possible to model every detail of the system, we still want to be able to drive consistent and insightful conclusions.

The fictitious coupling type $\{\sharp\}$ of L_a does not allow any



(a) Example Decision Spaces (b) Request Types Overlap

Fig. 1. Inter-Layer Composition

choice in the coupling space, hence we cannot distinguish different policy decision points in this layer. The reason why we do not consider multiple firewalls is that the interest in analyzing distributed network filtering typically concerns intra-layer dependencies, whereas we only want to model (and reason about) inter-layer dependencies. We believe that intra-layer reasoning is an orthogonal problem that, as argued in Section VI, has already been addressed in related work.

Note that more layers could be added on top of L_b. For instance, Web applications running on certain virtual hosts within the Web server and performing access control based on application-specific fields, such as users, roles, actions and resources. However, for the sake of conciseness, in this paper we limit the scope of our analysis to these two layers.

III. COMPOSITION AND DECOMPOSITION

In this section we define composition as a binary operation between ACLs. In order to compose access control layers, we first need a way to combine the decisions yielded by their respective policies. We thus briefly discuss how this is achieved by equipping the set of decisions with a suitable algebraic structure named *decision space*.

The standard decision space is $\mathbb{B} = \{0, 1\}$ where 0 stands for prohibition and 1 for authorization. Note that in this case the decision function $\delta : \mathfrak{Q}(F) \to \mathbb{B}$ simply tests whether some request $q \in \mathfrak{Q}(F)$ is a member of a set $Q^{Auth} \subseteq \mathfrak{Q}(F)$ of authorized queries. Many existing languages (e.g., [5], [6], [7]) assume that the decision space is larger than $\mathbb{B}$ to include for instance undefined decision ($\bot$) or conflicting decision ($\top$) to cope with modular specification of authorization policies. Here we equip the decision space with operators, denoted $\sqcup$ and $\sqcap$, that generalize the boolean disjunction and conjunction.

Definition 5 (Decision Space): A decision space is a bounded distributive lattice $\langle D, \sqcup, \sqcap \rangle$, where D is a non empty finite set of decisions and $\sqcup, \sqcap$ are respectively the least upper bound and greatest lower bound operators on D. The lattice top and bottom elements are denoted respectively $\mathbf{1}_D$ and $\mathbf{0}_D$.

Where no ambiguity arises we identify a decision space with its underlying set D. Figure 1a shows the Hasse diagrams of the boolean decision space $\mathbb{B}$ and its extension to undefined decisions $\mathbb{B}_\bot$, which we use throughout the rest of the paper.

The key to ACL composition is the overlap of request types, because it implies interdependency between decisions taken by different decision functions. Let $L_1 = \langle F_1, C_1, \delta_1 \rangle$ and $L_2 = \langle F_2, C_2, \delta_2 \rangle$ be two ACLs that act in composition, e.g.,

TABLE IV
EXAMPLE DECISION FUNCTION δ_c

$\mathsf{I_s}$	$\mathsf{I_d}$	$\mathsf{P_s}$	$\mathsf{P_d}$	H	U	D
1.1.1.∗	1.1.1.1	∗	≤79	∗	∗	⊥
1.1.1.∗	1.1.1.1	∗	80	acme.com	/public/∗	1
1.1.1.∗	1.1.1.1	∗	80	acme.com	/private/∗	1
1.1.1.∗	1.1.1.1	∗	80	beta.com	/∗	1
1.1.1.∗	1.1.1.1	∗	≥81	∗	∗	⊥
1.1.1.∗	1.1.1.∗\ 1.1.1.1	∗	∗	∗	∗	⊥
2.2.∗.∗	1.1.1.1	∗	80	acme.com	/public/∗	1
2.2.2.∗	1.1.1.1	∗	80	acme.com	/private/∗	1
2.2.3.∗	1.1.1.1	∗	80	beta.com	/∗	1
3.3.∗.∗	1.1.1.∗\ 1.1.1.1	∗	80	∗	∗	⊥
3.3.∗.∗\ 3.3.3.∗	1.1.1.1	∗	80	acme.com	/public/∗	1
		...				0

suppose L_2 is over L_1 in the network stack. Then, every lower layer request shall match to some upper layer policy decision point, hence the upper layer coupling type C_2 is included in the lower layer request type F_1. Furthermore, it may be the case that the two layers' request types have some fields in common (i.e., F_1 and F_2 have a non empty intersection). Figure 1b depicts this situation, where the double hatched areas highlight the overlap between layers.

The union of the request types of L_1 and L_2 can then be thought as the request type of a new ACL L, that we are going to define as their composition. The decision function of L needs to depend both on δ_1 and δ_2. If we assumed a boolean decision space, then we would expect every lower layer request q_l that agrees with an upper layer request q_u to yield a composite request that is allowed *if and only if* both q_l and q_u are allowed. In the following definition we generalize this intuition by substituting the logic conjunction by the greatest lower bound operator $\sqcap$.

Definition 6 (ACL Composition): Given the ACLs $L_1 = \langle F_1, C_1, \delta_1\rangle$ and $L_2 = \langle F_2, C_2, \delta_2\rangle$ such that $C_2 \subseteq F_1$ and $C_1 \cap F_2 = \emptyset$, their composition is the ACL $L_1 \otimes L_2 = \langle F_1 \cup F_2, C_1, \delta\rangle$, where δ is defined as follows:

$$\begin{aligned}\delta : \mathfrak{Q}(C_1 \cup F_1 \cup F_2) &\to D \\ q &\mapsto \delta_1(q|_{C_1\cup F_1}) \sqcap \delta_2(q|_{C_2\cup F_2}).\end{aligned} \tag{1}$$

As an example, we consider the composition L_c of ACLs L_a and L_b introduced in Section II. $L_a \otimes L_b = L_c = \langle\{\mathsf{I_s}, \mathsf{I_d}, \mathsf{P_s}, \mathsf{P_d}, \mathsf{H}, \mathsf{U}\}, \{\sharp\}, \delta_c\rangle$, where the graph of δ_c is represented in Table IV. For instance, the request ⟨2.2.2.1, 1.1.1.1, 0, 80, acme.com, /private/⟩ is authorized in L_c because L_a allows ⟨2.2.2.1, 1.1.1.1, 0, 80⟩ and L_b allows ⟨2.2.2.1, acme.com, /private/⟩. However, the decision related to the request ⟨3.3.3.3, 1.1.1.20, 0, 80, acme.com, /private/⟩ is ⊥ because it is unknown whether any Web server is listening on the IP address 1.1.1.20.

Given an ACL $L = \langle F, C, \delta\rangle$, decomposition is the problem of finding ACLs $L_1 = \langle F_1, C_1, \delta_1\rangle$ and $L_2 = \langle F_2, C_2, \delta_2\rangle$ such that $L_1 \otimes L_2 = L$. As Definition 6 fixes the request types of the candidate decompositions, the problem amounts to compute δ_1 and δ_2 from δ. This means that, for every request $q_1 \in \mathfrak{Q}(F_1 \cup C_1)$, we need the value of $\delta_1(q_1)$ as a function of all the requests $q \in \mathfrak{Q}(F)$ such that $q|_{F_1\cup C_1} = q_1$.

In the case of a boolean decision space, the natural semantics we would like to assign to such an operation is that of *projection*. For instance, let $\delta : \mathfrak{Q}(F) \to \mathbb{B}$. Its projection on $P \subseteq F$ would be $\pi_P(\delta) : \mathfrak{Q}(P) \to \mathbb{B}$ such that $\pi_P(\delta)(q) = 1$ if and only if $\delta(q') = 1$ for at least one $q' \in \mathfrak{Q}(F)$ s.t. $q'|_{F_1\cup C_1} = q$. Hence, $\pi_P(\delta)$ would map each q to the logic disjunction of all such $\delta(q')$. In the next definition we generalize to larger decision spaces by replacing disjunction with least upper bound $\sqcup$.

Definition 7 (Projection): The projection of a decision function $\delta : \mathfrak{Q}(F) \to D$ over the set of fields $P \subseteq F$ is the decision function $\pi_P(\delta)$ defined as follows:

$$\begin{aligned}\pi_P(\delta) : \mathfrak{Q}(P) &\to D \\ q &\mapsto \bigsqcup_{x\in\mathfrak{Q}(F\setminus P)} \delta(q + x).\end{aligned} \tag{2}$$

As stated in the next proposition, *decompositions* obtained through projection are always the *least permissive* among those that preserve the composite decision function unchanged.

Proposition 1 (Least Privilege Decomposition): Let $L = \langle F_1 \cup F_2, C_1, \delta\rangle$ such that $L = \langle F_1, C_1, \delta_1\rangle \otimes \langle F_2, C_2, \delta_2\rangle$. Then, $\forall q \in \mathfrak{Q}(F_i \cup C_i)$, $\pi_{F_i\cup C_i}(\delta)(q) \leq \delta_i(q)$ for $i \in \{1, 2\}$.

This result guarantees the least privilege principle for policy refactoring as stated in the introduction. We begin to see that the definitions of composition and decomposition can be suitable tools to tackle the refactoring problem.

IV. COMPACT REPRESENTATION

The request spaces we considered so far are, in general, infinite or very large in cardinality. This follows from the fact that each field can have either an infinite (e.g. URLs) or a very big (e.g. IP addresses) domain of values. In order to cope with this issue, in this section we introduce a finite and compact representation for generic request spaces and decision functions. We then show how (de)composition can be computed on instances of such a representation.

Definition 8 (Field Descriptor): Given a request field $f \in \mathcal{F}$, a field descriptor for f is a structure $\langle\Phi_f, [\![\cdot]\!]_f\rangle$ where Φ_f is a language that allows to describe sets of elements in the domain of f and $[\![\cdot]\!]_f : \Phi_f \to 2^{\mathrm{dom}(f)}$ is an interpretation function that maps every sentence of the language to its extension, i.e., the subset of the field's domain it describes.

Furthermore, the language Φ_f is closed under the intersection of sentences' extensions, namely $\forall\varphi_1, \varphi_2 \in \Phi_f$, $\exists\varphi_3 \in \Phi_f$ s.t. $[\![\varphi_3]\!]_f = [\![\varphi_1]\!]_f \cap [\![\varphi_2]\!]_f$. For every such combination we call φ_3 the conjunction of φ_1, φ_2, written $\varphi_3 = \varphi_1 \wedge \varphi_2$.

To illustrate the concept, we consider a field descriptor that allows to describe intervals of positive integers. A bounded interval is described by a pair of integers representing its minimum and maximum values; an unbounded one has its maximum value set to ∞. The conjunction of two intervals is the (potentially empty) interval ranging from the maximum of their lower boundaries to the minimum of their upper ones. E.g., if $\varphi_1 = [0, 100], \varphi_2 = [20, \infty]$ and $\varphi_3 = [200, 300]$, we have $\varphi_1 \wedge \varphi_2 = [20, 100]$ and $[\![\varphi_1 \wedge \varphi_3]\!] = \emptyset$, $[\![\varphi_1]\!] = \{0, 1, \ldots, 100\}$, $[\![\varphi_1 \wedge \varphi_2]\!] = \{20, 21, \ldots, 100\}$.

This descriptor is already suitable for representing many of the fields introduced in Table I (i.e., $\mathsf{I_s}, \mathsf{I_d}, \mathsf{P_s}, \mathsf{P_d}$). For fields associated with large or infinite domains of strings (such as U) we could analogously define a field descriptor where the sentences in Φ are arbitrary regular expressions, and their conjunction is the intersection of the corresponding automata.

When we put together a collection of field descriptors, we obtain an object suitable to describe sets of requests. This is formalized in the following definition, which recasts Definitions 2 and 3 to the language of field descriptors.

Definition 9 (Requests Descriptor): Let F be a set of fields and, for every field $f \in F$, let $\langle \Phi_f, [\![\cdot]\!]_f \rangle$ be an associated field descriptor. We then define the requests descriptors space on $F = \{f_i\}_{i=1}^n$ as the product of all the languages Φ_{f_i}: $\Psi(F) = \Phi_{f_1} \times \ldots \times \Phi_{f_n}$. Every sequence $\psi = \langle \varphi_1, \ldots, \varphi_n \rangle \in \Psi(F)$ is a Request Descriptor (RD) for the request space $\mathfrak{Q}(F)$.

The concatenation $\psi + \psi'$ and the projection $\psi|_{P \subseteq F}$ from Definition 3 extend naturally to RDs. Moreover, if $\psi = \langle \varphi_1, \ldots, \varphi_n \rangle, \psi' = \langle \varphi'_1, \ldots, \varphi'_n \rangle$ are RDs on $\mathfrak{Q}(F)$, we define their conjunction as $\psi \wedge \psi' = \langle \varphi_1 \wedge \varphi'_1, \ldots, \varphi_n \wedge \varphi'_n \rangle$.

The extension of a RD ψ, written $[\![\psi]\!]_F$, is the product of the extension of the sentences φ_i. Formally: $[\![\psi]\!]_F = [\![\psi(f_1)]\!]_{f_1} \times \ldots \times [\![\psi(f_n)]\!]_{f_n}$.

Now, we can define a finite descriptor for decision functions.

Definition 10 (Decision Function Descriptor): Given a set of fields F and a decision space D, a Decision Function Descriptor (DFD) is a finite relation $\Delta \subseteq \Psi(F) \times D$ that covers the entire request space $\mathfrak{Q}(F)$.

The extension of Δ is the decision function $\delta : \mathfrak{Q}(F) \to D$, written $\delta = \mathrm{ext}(\Delta)$, defined as follows:

$$\delta(q) = \bigsqcup \{d \mid \langle \psi, d \rangle \in \Delta \wedge q \in [\![\psi]\!]_F\}, \forall q \in \mathfrak{Q}(F). \quad (3)$$

Equation (3) associates each DFD with a unique decision function (namely its extension), which can be thought of as its semantics. The DFD extension maps every request q to the least upper bound of all the decisions being associated, in the DFD, with a request descriptor that matches q. Note how, given this semantics, requiring the complete coverage of the entire request space is not restrictive. In fact, this can always be achieved by including in the DFD a *default* RD that (i) matches all the possible requests and (ii) is associated with the decision $\mathbf{0}_D$. Moreover, whenever a concrete policy language features a *deny by default* semantics (as it is typically the case, e.g., for firewalls), the translation of such policies to DFD reduces to computing decisions for all the possible overlaps among rules within the policy. As discussed in Section VI the latter problem has been already extensively studied in literature on policy conflicts, hence it is left out in this paper.

Algorithm 1 defines two procedures on DFDs that are consistent with the (de)composition of the respective extensions. The correctness of the algorithm, as stated in Proposition 2, is ensured by showing that the extension of the output DFDs equals the composition (Definition 6), resp. projection (Definition 7), of the input ones.

Proposition 2 (Correctness of Algorithm 1): When $\textsc{DFDComp}(\langle F_1, C_1, \Delta_1 \rangle, \langle F_2, C_2, \Delta_2 \rangle) = \langle F_1 \cup F_2, C_1, \Delta \rangle$, *then* $\langle F_1, C_1, \mathrm{ext}(\Delta_1) \rangle \otimes \langle F_2, C_2, \mathrm{ext}(\Delta_2) \rangle = \langle F_1 \cup F_2, C_1, \mathrm{ext}(\Delta) \rangle$. *Moreover, when* $\textsc{DFDProj}(\Delta) = \Delta'$, *then* $\pi_P(\mathrm{ext}(\Delta)) = \mathrm{ext}(\Delta')$.

Algorithm 1 ACL Composition and Projection with DFD

```
1: procedure DFDComp(⟨F1, C1, Δ1⟩, ⟨F2, C2, Δ2⟩)
2:     W ← (F1 ∩ F2) ∪ C2
3:     U ← (F1 ∪ C1) \ W
4:     V ← (F2 ∪ C2) \ W
5:     Δ ← ∅
6:     for all ⟨ψ1, d1⟩ ∈ Δ1, ⟨ψ2, d2⟩ ∈ Δ2 do
7:         if [[ψ1|W ∧ ψ2|W]]W ≠ ∅ then
8:             ψ ← (ψ1|W ∧ ψ2|W) + ψ1|U + ψ2|V
9:             Δ ← Δ ∪ {⟨ψ, d1 ⊓ d2⟩}
10:        end if
11:    end for
12:    return ⟨F1 ∪ F2, C1, Δ⟩
13: end procedure
14:
15: procedure DFDProj(Δ, P)
16:    return {⟨ψ|P, d⟩ | ⟨ψ, d⟩ ∈ Δ}
17: end procedure
```

Note how Algorithm 1 benefits from the choice of allowing the descriptors in a DFD to overlap. In particular, the operations are defined without ever computing the disjunction or the complement of RDs (resp. the union and the difference of their extensions). This is convenient as the Cartesian product distributes with respect to intersection, but not to union and difference. Hence, as argued in [8], a number of RDs that grows linearly with the dimension of the product would be necessary to represent each union or difference.

In order to perform refactoring, we need to determine whether a decomposition is possible. This is equivalent to check if a decision function, once projected and composed back, equals itself. This translates into testing the equivalence of $\langle F_1 \cup F_2, C_1, \Delta \rangle$ with $\langle F_1, C_1, \pi_{F_1 \cup C_1}(\Delta) \rangle \otimes \langle F_2, C_2, \pi_{F_2 \cup C_2}(\Delta) \rangle$, which, as different DFDs can have the same extension, means to compare the (possibly infinite) extensions of their DFDs. In the next section we deal with this issue by developing an alternative criterion to test decomposability.

V. Refactoring

Through decomposition, we aim at factorizing the complexity of some layer's policy into simpler ones. This means that the request type of any of the decomposed layers shall be a strict subset of the one of the original (composite) layer.

The next result shows that it is not guaranteed that such a decomposition exists for a generic access control layer.

Proposition 3: Given an ACL $L = \langle F_1 \cup F_2, C_1, \delta \rangle$ it is not always possible to find two ACLs $\langle F_1, C_1, \delta_1 \rangle$, $\langle F_2, C_2, \delta_2 \rangle$ that decompose L with $F_2 \not\subseteq F_1$.

The last result can be illustrated through the following counterexample. Consider the decision function δ_c (Table IV) and let $q_1 = \langle$2.2.2.1, 1.1.1.1, 0, 80, acme.com, /private/$\rangle$, $q_2 = \langle$2.2.3.1, 1.1.1.1, 0, 80, acme.com, /private/$\rangle$ and $q_3 = \langle$2.2.3.1, 1.1.1.1, 0, 80, beta.com, /$\rangle$. Note that, in case $C_1 \cup F_1 = \{\mathsf{I_s}, \mathsf{I_d}, \mathsf{P_s}, \mathsf{P_d}\}$ and $C_2 \cup F_2 = \{\mathsf{I_d}, \mathsf{P_d}, \mathsf{H}, \mathsf{U}\}$, we have indeed a contradiction. In fact, as $\delta(q_2) = 0$, we

need either $\delta_1(\langle 2.2.3.1,\ 1.1.1.1,\ 0,\ 80\rangle) = \delta_1(x) = 0$ or $\delta_2(\langle \text{acme.com},\ /\text{private}/\rangle) = \delta_2(y) = 0$. On the other hand, as $\delta(q_1) = \delta(q_3) = 1$, both $\delta_1(x) = 1$ and $\delta_2(y) = 1$ must hold. Intuitively, we see that in order to have decomposability, the decisions associated to requests that satisfy a specific inter-field dependency cannot be chosen independently one from another. The next definition formalizes this intuition.

Definition 11 (Inter-Field Dependency): For W and V non-empty and disjoint subsets of F, we say that a decision function δ satisfies the Inter-Field Dependency (IFD) condition $W \twoheadrightarrow V$, written $\delta \models W \twoheadrightarrow V$, if and only if $\forall q, q' \in \mathfrak{Q}(F),\ q|_W = q'|_W \Rightarrow \delta(q) \sqcap \delta(q') = \delta(q|_{F\setminus V} + q'|_V) \sqcap \delta(q|_V + q'|_{F\setminus V})$.

We are now ready to state the main result of this section: IFDs precisely characterize when an ACL can be decomposed by projections without loss.

Theorem 4 (Decomposability): Given a generic ACL $L = \langle F_1 \cup F_2, C_1, \delta\rangle$, the following are equivalent:

- $L = \langle F_1, C_1, \pi_{F_1\cup C_1}(\delta)\rangle \otimes \langle F_2, C_2, \pi_{F_2\cup C_2}(\delta)\rangle$
- $\delta \models (F_1 \cap F_2) \cup C_2 \twoheadrightarrow (F_2 \setminus F_1)$.

Theorem 4 gives an alternative criterion to test the decomposability of an ACL: we need to check if its decision function satisfies the IFD $(F_1 \cap F_2) \cup C_2 \twoheadrightarrow (F_2 \setminus F_1)$, for the subsets F_1, F_2, C_2 of its request type (with $C_2 \subseteq F_1$) that represent the new layers' layout we want to find a policy for.

IFDs are particularly interesting because they are constraints on the internal structure of the decision function that we can as well check on any corresponding DFDs. Algorithm 3 computes this check and the next proposition ensures its correctness.

Proposition 5 (Correctness of Algorithm 3): Let $\Delta \subseteq \Psi(F) \times D$ be a DFD and W, V two non-empty and disjoint subsets of F. Then, $\Delta \models W \twoheadrightarrow V \Leftrightarrow \text{ext}(\Delta) \models W \twoheadrightarrow V$.

The key ideas underlying Algorithm 3 are as follows. First, we refactor the RDs contained in Δ to make them partition the entire request space (lines 2–7) such that every request matches exactly one RD $\psi_W + \psi_U + \psi_V$. Second, for every ψ_W, we consider all the pairs ψ_U, ψ'_U and ψ_V, ψ'_V (lines 8, 9) and we compute the greatest lower bound of the decisions associated with all the pairs of requests matching respectively $\psi_W + \psi_U + \psi_V$ and $\psi_W + \psi'_U + \psi'_V$ (line 10). We finally check if the latter equals the greatest lower bound of all the pairs of requests matching $\psi_W + \psi_U + \psi'_V$ and $\psi_W + \psi'_U + \psi_V$ (lines 11–18). To do the request space partition, we use iteratively the procedure defined by Algorithm 2 that computes a closure w.r.t. intersection and difference for the portion of the RDs concerning the subset of fields X. Note that here, unlike for previous algorithms, we need to compute the set of RDs describing the difference between two given RDs, formally $\textsc{Diff}(\psi_1, \psi_2) = \{\psi_i^*\}$ s.t. $\bigcup\{[\![\psi_i^*]\!]_F\} = [\![\psi_1]\!]_F \setminus [\![\psi_2]\!]_F$. This is generally possible for the RDs composed of the field descriptors considered in this paper (e.g., intervals of integers).

Given any ACL we can now first test for decomposability by the means of Algorithm 3 and then, in case of success, project to obtain the desired decomposition.

Consider, for example, the ACL $L_3 = \langle\{\mathsf{I_s}, \mathsf{I_d}, \mathsf{P_s}, \mathsf{P_d}, \mathsf{H}, \mathsf{U}\}, \{\sharp\}, \delta_c\rangle$ that was introduced in Section III. As it is the result of the composition of ACLs $L_a = \langle\{\mathsf{I_s}, \mathsf{I_d}, \mathsf{P_s}, \mathsf{P_d}\}, \{\sharp\}, \delta_a\rangle$ and $L_b = \langle\{\mathsf{I_s}, \mathsf{H}, \mathsf{U}\}, \{\mathsf{I_d}, \mathsf{P_d}\}, \delta_b\rangle$, we naturally expect it to be decomposable with respect to their same structure. The reader can check, by inspecting Table IV, that indeed $\delta_c \models \{\mathsf{I_s}, \mathsf{I_d}, \mathsf{P_d}\} \twoheadrightarrow \{\mathsf{H}, \mathsf{U}\}$. Hence, if we name $L'_a = \langle\{\mathsf{I_s}, \mathsf{I_d}, \mathsf{P_s}, \mathsf{P_d}\}, \{\sharp\}, \pi_{\{\mathsf{I_s}, \mathsf{I_d}, \mathsf{P_s}, \mathsf{P_d}\}}(\delta_c)\rangle$ and $L'_b = \langle\{\mathsf{I_s}, \mathsf{H}, \mathsf{U}\}, \{\mathsf{I_d}, \mathsf{P_d}\}, \pi_{\{\mathsf{I_s}, \mathsf{I_d}, \mathsf{P_d}, \mathsf{H}, \mathsf{U}\}}(\delta_c)\rangle$, we know by Theorem 4 that $L_3 = L'_a \otimes L'_b$. This is an example of refactoring that keeps the request type unchanged.

Table V represents the decision function $\pi_{\{\mathsf{I_s}, \mathsf{I_d}, \mathsf{P_s}, \mathsf{P_d}\}}(\delta_c)$. Notice that it is not exactly equal to the original decision function δ_a of the ACL L_a (cf. Table II). In particular, it is *never more permissive* than the original; on the other hand it is, where possible and according to the least privilege principle, *more restrictive*. For instance, requests coming from

Algorithm 2 DFD Partition

Input: $\Delta \subseteq \Psi(F) \times D$.
Input: $X \subseteq F$.
Output: $\textsc{Partition}(\Delta, X)$
1: $Y \leftarrow F \setminus X$
2: $\mathcal{P} \leftarrow \emptyset$
3: **for all** $\langle\psi, d\rangle \in \Delta$ **do**
4: $\quad\mathcal{P} \leftarrow \mathcal{P} \cup \{\psi|_X, \{\langle\psi|_Y, d\rangle\}\}$
5: **end for**
6: **while** $\exists\, \langle\psi_1, \Delta_1\rangle, \langle\psi_2, \Delta_2\rangle \in \mathcal{P}$ s.t. $[\![\psi_1 \wedge \psi_2]\!]_X \neq \emptyset$ **do**
7: $\quad\mathcal{P} \leftarrow \mathcal{P} \setminus \{\langle\psi_1, \Delta_1\rangle, \langle\psi_2, \Delta_2\rangle\}$
8: $\quad\mathcal{P} \leftarrow \mathcal{P} \cup \{\langle\psi_1 \wedge \psi_2, \Delta_1 \cup \Delta_2\rangle\}$
9: $\quad$**for all** $\psi_{1\setminus 2} \in \textsc{Diff}(\psi_1, \psi_2), \psi_{2\setminus 1} \in \textsc{Diff}(\psi_2, \psi_1)$ **do**
10: $\quad\quad$**if** $[\![\psi_{1\setminus 2}]\!]_X \neq \emptyset$ **then**
11: $\quad\quad\quad\mathcal{P} \leftarrow \mathcal{P} \cup \{\langle\psi_{1\setminus 2}, \Delta_1\rangle\}$
12: $\quad\quad$**end if**
13: $\quad\quad$**if** $[\![\psi_{2\setminus 1}]\!]_X \neq \emptyset$ **then**
14: $\quad\quad\quad\mathcal{P} \leftarrow \mathcal{P} \cup \{\langle\psi_{2\setminus 1}, \Delta_2\rangle\}$
15: $\quad\quad$**end if**
16: $\quad$**end for**
17: **end while**
18: **return** $\mathcal{P}$

Algorithm 3 Inter-Field Dependency check on DFD

Input: $\Delta \subseteq \Psi(F) \times D$.
Input: $W \twoheadrightarrow V$, with $W, V \in 2^F \setminus \{\emptyset\}$, $W \cap V = \emptyset$.
Output: true/false
1: $U \leftarrow F \setminus (W \cup V)$
2: **for all** $\langle\psi_W, \Delta_W\rangle \in \textsc{Partition}(\Delta, W)$ **do**
3: $\quad\mathcal{P}_U, \mathcal{P}_V \leftarrow \emptyset$
4: $\quad$**for all** $\langle\psi_U, \Delta_U\rangle \in \textsc{Partition}(\Delta_W, U)$ **do**
5: $\quad\quad\mathcal{P}_U \leftarrow \mathcal{P}_U \cup \{\psi_U\}$
6: $\quad\quad\mathcal{P}_V[\psi_U] \leftarrow \textsc{Partition}(\Delta_U, V)$
7: $\quad$**end for**
8: $\quad$**for all** $\psi_U, \psi'_U \in \mathcal{P}_U$ **do**
9: $\quad\quad$**for all** $\langle\psi_V, \Delta_V\rangle \in \mathcal{P}[\psi_U], \langle\psi'_V, \Delta'_V\rangle \in \mathcal{P}[\psi'_U]$ **do**
10: $\quad\quad\quad d_1 \leftarrow \bigsqcup\{d \mid \langle\cdot, d\rangle \in \Delta_V\} \sqcap \bigsqcup\{d \mid \langle\cdot, d\rangle \in \Delta'_V\}$
11: $\quad\quad\quad$**for all** $\langle\psi''_V, \Delta''_V\rangle \in \mathcal{P}[\psi_U], \langle\psi'''_V, \Delta'''_V\rangle \in \mathcal{P}[\psi'_U]$ **do**
12: $\quad\quad\quad\quad$**if** $[\![\psi'_V \wedge \psi''_V]\!]_V \neq \emptyset \wedge [\![\psi_V \wedge \psi'''_V]\!]_V \neq \emptyset$ **then**
13: $\quad\quad\quad\quad\quad d_2 \leftarrow \bigsqcup\{d \mid \langle\cdot, d\rangle \in \Delta''_V\} \sqcap \bigsqcup\{d \mid \langle\cdot, d\rangle \in \Delta'''_V\}$
14: $\quad\quad\quad\quad\quad$**if** $d_1 \neq d_2$ **then**
15: $\quad\quad\quad\quad\quad\quad$**return false**
16: $\quad\quad\quad\quad\quad$**end if**
17: $\quad\quad\quad\quad$**end if**
18: $\quad\quad\quad$**end for**
19: $\quad\quad$**end for**
20: $\quad$**end for**
21: **end for**
22: **return true**

TABLE V
EXAMPLE PROJECTION $\pi_{\{\mathsf{I_s},\mathsf{I_d},\mathsf{P_s},\mathsf{P_d}\}}(\delta_c)$

$\mathsf{I_s}$	$\mathsf{I_d}$	$\mathsf{P_s}$	$\mathsf{P_d}$	D
1.1.1.∗	1.1.1.1	∗	$\leq$79	$\perp$
1.1.1.∗	1.1.1.1	∗	80	1
1.1.1.∗	1.1.1.1	∗	$\geq$ 81	$\perp$
1.1.1.∗	1.1.1.∗\ 1.1.1.1	∗	∗	$\perp$
2.2.∗.∗	1.1.1.1	∗	80	1
3.3.∗.∗\ 3.3.3.∗	1.1.1.1	∗	80	1
3.3.∗.∗	1.1.1.∗\ 1.1.1.1	∗	80	$\perp$
	...			0

the 3.3.3.0/24 IP network and directed to 1.1.1.1, which were allowed by the original policy, are denied by the refactored version. This is consistent with the fact that such requests were anyway already denied in L_b (cf. Table III). On the other hand, the decisions for all requests having source within 3.3.0.0/24 and directed to the rest of the 1.1.1.0/24 network, are refactored to $\perp$. This is a consequence of assuming partial knowledge of the L_b policy. Since additional information is required to decide on the usefulness of those permissions, the choice is left to the user who can either trust the original policy, i.e., change $\perp$ to 1, or follow a more restrictive approach and change $\perp$ to 0. Let us now try to refactor L_a, L_b to a pair of ACLs that have smaller request types. Suppose, for instance, to substitute the `WS` Web server of our scenario with one that does not discriminate requests on the basis of the IP source address; this means that the field $\mathsf{I_s}$ does not belong any more to the request type of L_b''. However, as shown by the counterexample given earlier to illustrate Proposition 3, we know that such a decomposition is not possible. This is indeed because $\delta_c \not\models \{\mathsf{I_d}, \mathsf{P_d}\} \twoheadrightarrow \{\mathsf{H}, \mathsf{U}\}$.

Had all the requests with $\mathsf{H} =$ acme.com in δ_c been mapped to 0, the IFD would have been instead satisfied. In such a case we would have had a refactoring with a change in request types that simplified the decision function δ_b.

VI. RELATED WORK

The pioneering work of Moffett and Sloman [9] has opened a large avenue for research on policy conflict analysis in distributed systems and has been refined, classified and formalized in the network-level security field. For instance, Al-Shaer *et al.* [10] or Basile *et al.* [8] have proposed techniques and algorithms to automatically discover and manage inconsistencies between firewall rule sets. One key point of these techniques is to turn rule sets into an intermediate policy representation on which analysis is performed (e.g., ordered binary decision diagram [10], intersection closed semi-lattice of subsets [8]).

Although this paper does not focus on inconsistencies, it shares the abstract definition of devices and the ideas behind composition of policies to capture and analyze interactions between different layers. Related work on network-level analysis may complete our approach by providing means to deal with interactions between firewalls.

There exists a substantial body of work on combination of policies not limited to firewalls. Notably, several logical and algebraic approaches for composing and unifying access control policies have been proposed quite recently [7], [5], [6]. Different algebraic varieties have been used to combine rich decision spaces, for instance, $\mathcal{D}$-algebras [7], Belnap bilattices [6] or XACML tailored logic [5]. One of the goals is to provide mathematical foundations to the expressive XACML access control language and its many resolution strategies.

The algebraic structure chosen here for decision space is motivated by previous work which have shown the need for expressive ones. However, the goal is not to capture resolution strategies but to find a structure both expressive enough and sufficient to define a decomposition with good properties. The use of an expressive common pivot model like XACML, logical frameworks or subject-target-condition rules [11] is very interesting. However, this paper chooses a different perspective by sticking as closely as possible to the original policies. Algorithms 1 and 3 carry computation directly on original policies, with minimal prior normalization.

Our approach has a strong connection to the policy continuum model and the policy refinement problem. Davy et al. [12] model policies at different inter-related abstraction layers in what they call policy continuum. Based on this model, they devise a generic algorithm for policy authoring. Their notion of continuum level essentially corresponds to a view on the policies at a particular abstraction level. In contrast, our ACLs represent types of decision points that operate at different architectural layers, but being at the same degree of abstraction (which roughly matches to the lowest possible continuum level). A similar argument applies to many existing works on policy refinement [13], [14], [11]. Another key difference is that we start from interdependent concrete policies and we provide a device oriented decomposition instead of starting from high level requirements which are ultimately refined into operational policies. The refactoring problem studied in this paper is *bottom-up*: the global policy is nothing but the composition of all devices interacting along the network stack. By contrast, the policy refinement problem is clearly *top-down*.

Finally, the definition of access control layer is quite close to that of abstract access control systems [15], [16], [4]. The first two references formally compare the expressiveness of access control models with respect to the set of decision functions they can produce. Interest is not brought on the state but on the model itself, whereas we focus on the first. For the last, a set of desirable properties of abstract access control systems is identified. The question whether our decomposition technique still applies in their case is left for future work.

VII. CONCLUSION

This paper proposes a relational oriented approach for access control policies decomposition. The key concept that captures decomposability is the inter-field dependency condition given in Section V from which an algorithm is derived. This algorithm works directly on a relational representation of access control policies. Relational database experts may have recognized the syntactical resemblance between $W \twoheadrightarrow V$ of Definition 11 and so-called *MultiValued Dependencies* (MVD) introduced in [3]. The two concepts are related in the fact that

Theorem 1 of [3], linking lossless decomposition to MVDs, is a specific instance of Theorem 4 of this paper, when the decision space is reduced to the two-elements boolean algebra.

We shall now discuss the main issues related to a potential implementation of our approach and conclude with an outlook on future research directions.

In order to execute our algorithms, input policies need to be represented as DFDs. We argued that, for rule-based access control languages where rules are set of independent filters on fields, this can be done by leveraging existing techniques. Other than most firewall configuration languages, the access control policies of many common network services fit into this category, arguably because of its simplicity. For those that do not, our approach can still be applied as long as the translation from the source language to DFDs remains feasible.

Further scalability issues are expected to be instead related to algorithmic complexity. Other than the size of input DFDs, we expect the degree of overlap between the RDs therein contained to play a critical role, in particular for what concerns Algorithm 2 that clearly performs worst when every RD has a non-empty intersection with any other. We plan to test our hypotheses by conducting a thorough experimental evaluation.

Finally, it is important to explore up to which scale the results of our analysis can still be consumable and insightful for users. This point is particularly critical when non-decomposability occurs. We believe that equipping the implementation with a root-cause analysis feature – which, e.g., isolates the fragments of the policies preventing decomposition – would improve significantly the usability of our approach.

As ongoing and future research work, we outline two more related problems for which we argue that techniques inspired from relational databases theory can be fruitfully applied.

The first problem is about repairs. Assume an access control policy over several fields cannot be decomposed into independent sub-policies: is there any canonical or best way to update the policy such that it will become decomposable? For instance, in the example presented at the end of Section V, one such possible modification is proposed for the policy δ_c. Applying the many results for the repair problems (e.g., see [17]) to policies is not without difficulties. Basically, one has to generalize the relational framework to be able to capture generic definition of access control systems and thus has to provide new results inspired from classical ones, as we have done for MVD in some sense. Moreover, one has to find repair semantics meaningful from the security point of view. Standard repair semantics are repair via insertions, deletions or updates [18]. It is interesting to study how these semantics apply to policy decomposition, the last one in particular.

The second problem is about the mining of dependencies. In the classical setting of database normalization, the set of dependencies is known in advance and one tries to provide the best structure fitting with the given constraints, as it is done in this paper. The data-mining perspective reverses the approach: The goal is to discover new dependencies by revealing the internal structure of relations. Savnik and Flach have provided an efficient algorithm to discover MVD [19], we envision to apply their techniques to reverse engineer large policies. The idea is to find some "best set of simplest access control systems" with the same authorized queries.

ACKNOWLEDGMENT

The authors would like to thank Henrik Plate and Mohand-Said Hacid for the useful technical advices and discussions as well as the anonymous reviewers for their comments that helped improve the manuscript.

This work was partially supported by the FP7-ICT-2009.1.4 Project PoSecCo (no. 257129, www.posecco.eu).

REFERENCES

[1] 7Safe and the University of Bedfordshire, "Uk security breach investigations report 2010," http://www.7safe.com/breach_report/Breach_report_2010.pdf, 2010.

[2] E. Al-Shaer, "Security automation research: Challenges and future directions," *IAnewsletter*, vol. 14, no. 4, pp. 14–18, 2011.

[3] R. Fagin, "Multivalued dependencies and a new normal form for relational databases," *ACM Trans. Database Syst.*, vol. 2, no. 3, pp. 262–278, Sep. 1977.

[4] J. Crampton and C. Morisset, "Towards a generic formal framework for access control systems," *CoRR*, vol. abs/1204.2342, 2012.

[5] C. D. P. K. Ramli, H. R. Nielson, and F. Nielson, "The logic of xacml," in *FACS*, 2011, pp. 205–222.

[6] G. Bruns and M. Huth, "Access control via belnap logic: Intuitive, expressive, and analyzable policy composition," *ACM Trans. Inf. Syst. Secur.*, vol. 14, no. 1, pp. 9:1–9:27, Jun. 2011.

[7] Q. Ni, E. Bertino, and J. Lobo, "D-algebra for composing access control policy decisions," in *ASIACCS '09: Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*. New York, NY, USA: ACM, 2009, pp. 298–309.

[8] C. Basile, A. Cappadonia, and A. Lioy, "Network-level access control policy analysis and transformation," *Networking, IEEE/ACM Transactions on*, vol. 20, no. 4, pp. 985–998, 2012.

[9] J. D. Moffett and M. S. Sloman, "Policy conflict analysis in distributed system management," *Journal of Organizational Computing*, vol. 4, pp. 1–22, 1994.

[10] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan, "Conflict classification and analysis of distributed firewall policies," *IEEE Journal on Selected Areas in Communications*, vol. 23(10), pp. 2069–2084, 2005.

[11] H. Zhao, J. Lobo, A. Roy, and S. M. Bellovin, "Policy refinement of network services for MANETs," in *The 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011)*, Dublin, Ireland, May 2011, to appear.

[12] S. Davy, B. Jennings, and J. Strassner, "The policy continuumpolicy authoring and conflict analysis," *Computer Communications*, vol. 31, no. 13, pp. 2981 – 2995, 2008.

[13] R. Craven, J. Lobo, E. C. Lupu, A. Russo, and M. Sloman, "Decomposition techniques for policy refinement," in *CNSM*, 2010, pp. 72–79.

[14] R. Craven, J. Lobo, E. Lupu, A. Russo, and M. Sloman, "Policy refinement: Decomposition and operationalization for dynamic domains," in *CNSM*, 2011, pp. 1–9.

[15] M. V. Tripunitara and N. Li, "A theory for comparing the expressive power of access control models," *J. Comput. Secur.*, vol. 15, no. 2, pp. 231–272, 2007.

[16] L. Habib, M. Jaume, and C. Morisset, "Formal definition and comparison of access control models," *Journal of Information Assurance and Security*, vol. 4, pp. 372–378, 2009.

[17] L. E. Bertossi, *Database Repairing and Consistent Query Answering*, ser. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[18] J. Wijsen, "Database repairing using updates," *ACM Trans. Database Syst.*, vol. 30, no. 3, pp. 722–768, Sep. 2005.

[19] I. Savnik and P. A. Flach, "Discovery of multivalued dependencies from relations," *Intell. Data Anal.*, vol. 4, no. 3,4, pp. 195–211, Sep. 2000.