Defense in Depth Formulation and Usage in Dynamic Access Control

Ridha Khedri[1]*, Owain Jones[2], and Mohammed Alabbad[1]

[1] Department of Computing and Software,
Faculty of Engineering, McMaster University
Hamilton, Ontario Canada
{khedri, alabbama}@mcmaster.ca
[2] CMC Microsystems,
Kingston, Ontario Canada
owain.jones@cmc.ca

Abstract. Many network systems secure their resources using a defense in depth strategy, which can result in complex policies being distributed on the many access control points of a network. These policies are subject to frequent changes to deal with different factors such as change in security situation or change in resources. Moreover, while we have a vague intuitive understanding of the defense in depth strategy, we certainly lack a rigorous definition for it that would allow us to objectively assess whether a policy distribution on a network satisfies this strategy. In this paper, we propose a definition for defense in depth based on a notion of refinement given in product family algebra. We use this definition to articulate several implementations of the defense in depth strategy taking into account local access policies and global constraints on the resources of the considered network. We also discuss the automation of the calculations needed to derive the appropriate access policies to deploy at the nodes of a network.

Keywords: Access Control Policies, Dynamic Access Control, Defense in Depth Strategy, Formal methods, Software Product Families, Algebraic Approaches

1 Introduction

Access control policies are a necessary tool toward mitigating security risks of network-accessible resources. They aim at protecting data and resources against unauthorized users, which contributes to ensuring information confidentiality and proper use of resources. When access control policies are comprehensive and well implemented, they shield the network system by creating a filter that restricts the access to only authorized users. An access control policy defines the (high-level) rules according to which access control must be regulated [24]. Many

* This research is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) through the grant RGPIN 2014-06115

policies need to include additional rules expressing the general security situation such as excluding some sources that are known to be facilitating or participating in building threats to resources. Moreover, real world network systems have more complex policies, where access decisions at firewalls depend on the application of different access rules coming from several sources and covering different societal and organizational perspectives (e.g., laws, practices, and organizational regulations). These policies are subject to frequent changes to deal with unexpected factors related to the general security situation or to changes in the service provided to users. Therefore, in an organization, access policies to resources involve different kinds of stakeholders that each bring a unique perspective on the conditions enabling access to resources. For instance, we can consider the views of management, finance, resource owners, and security officers as examples of relevant views in articulating access policies. Commonly, the policies derived from these views share some rules and differ on others. Hence, from this perspective, we can consider that we have a family of policies that have commonality and variability. The actual policy executed on a firewall is derived from these policies; most current firewalls execute a policy that is a sequential composition of these. One can conceive firewalls that execute these policies in parallel if the policies are composed of rules that satisfy the integrability property (i.e., consistency property) presented on Page 11.

The adaptation of networks to cope with changing security factors is often performed manually. In addition, different variants of access control policies need to be systematically integrated. In integrating them and then distributing them on the several firewalls of the network, or to different access control points, one needs to take into account their commonality and variability. From this perspective, the overall network access policy can be looked at as a family of policies where the members of the families might have similarity and slight variability. Taking this view demands a product line engineering approach for enhanced reuse of policies and factoring common access policies to low nodes in the network to be applied at firewalls closer to the perimeter of the access-target resource.

To reason on policies and to amend defense mechanisms on the fly require the automation of the reasoning induced by changes to the security situation and the communication of the attained decisions to concerned access control nodes. The automation is critical when we are considering a large network with a considerable number of dynamic resources (created and removed as needed such as in the case of virtual machines). The goal for the reasoning task is to ensure the consistency among policies (integrability as it is addressed in our work) or to determine the best way to assign policies to nodes. These needs have been pointed to by Burns et al. in [1] more than a decade ago. However, the progress remains very slim in attaining this goal. In [28], Dave Clark states:

> The idea that people are still programming routers using CLIs is a little mind-boggling. And the very idea that human beings are expected to figure out the global consequences of what might happen if they should

> make one little fix here or another little fix there... it's like we never escaped the 1980s!

The proposed work provides the background for reasoning about security policies towards automatic and dynamic defense mechanisms. At this stage of our work, and with the collaboration of our industrial partner CMC Microsystems, we developed a prototype tool that performs the calculations needed for the verification of the integrability of policies (i.e., consistency among policies) and for assigning to each node its policy that satisfies, according to Propositions 3–5, the Defense in Depth (DD) strategy.

DD strategy aims to defend valuable assets by creating layers of defenses that challenge the attacker in attaining access to the protected assets. Also, it is a strategy that calls for the network to be aware and self-protective. It has showed its merit in several areas such as fire prevention or nuclear energy. It is also intuitively used in [18] for network security using access policies. In our context of network resources protection, it has to put the resources behind layers of defensive policies that are more and more deterministic in the actions they take and the permission they grant. There are several basic questions that rise in our context. The most pressing questions are the following: How can we formally articulate this security strategy? If we are given a network topology and the policies assigned to each of its firewalls, how can we assess whether indeed we have our policies assigned according to the DD strategy? Are there schemes for assigning policies to access nodes that lead to a network of access control points employing DD strategy? In the remainder of this section, we are going to tackle these questions.

Another aspect to the problem is related to one of the fundamental tenants of secure designs. It is about not relying on one policy to achieve security, nor to locate all your policies in one access point. Multiple independent access points enabling access policies should be employed assuring a defense in depth [27]. However, an unauthorized user should be kept as far from the resource as possible. They should be blocked by the outermost possible firewall on the path to the sought resource. The proposed approach allows us to use algebraic calculations to determine the common policies that deny user access and then assign the role of denying them access to the outermost possible firewall.

The paper approaches the problem of assigning access control policies to firewalls from a product family perspective. It uses Product Family Algebra (PFA) to reason on policies within an information system as a family of related policies. Then by modelling access control rules as guarded commands, they can detect conflict among rules assigned to a firewall. The paper proposes for the first time a formal definition to the DD strategy. As far as we know, DD has been discussed only intuitively in the literature. We then propose several schemes for deploying policies according to the DD strategy. Also, PFA algebraic calculations enable us to determine the exact set of rules to be assigned to the firewall.

In Section 2, we give the background needed to make the paper self-contained. In Subsection 2.1, we briefly present the various access control policies found in the literature. In Subsection 2.2, we present PFA and guarded commands and

their mathematics. In Section 3, we formally articulate our understanding of DD strategy. Then, we propose schemata to assign access policies to nodes in order to get a network that employs a DD strategy. In Section 4, we discuss the automation of the proposed approach to implementing DD strategy and we describe the architecture of the prototype tool we are using. In Section 5, we discuss the merit of adopting a product family approach to reason on access control security and what would be the contexts where this paradigm can be helpful. We also, assess the strengths of our approch and its limitations. Through these limitations, we point to future research work. In Section 6, we briefly recap the main results of the paper.

2 Background

2.1 Access Control

An Access Control List (ACL) is the most basic form of access control specification. A resource on a system to which access should be controlled by an ACL is referred to as an object. We find also that we have Role-Based Access Control (RBAC) [5,6,25,26] in which access rests on the requester's role or function. When the decisions to access resources are based on a set of characteristics, or attributes, associated with the requester, the environment, and/or the resource itself, we have Attribute Based Access Control (ABAC) [14,15]. Each attribute is a field in a session state that a policy decision point can compare against a set of values to reach a decision on the appropriate action to take regarding access to the requested resource. When we take into account the dynamic nature of the security situation and would like to have realtime, adaptable, risk-aware access control to the enterprise, we have what is referred to as Risk-Adaptive Access Control (RAdAC) [2].

Current (hardware) firewalls implement either ABAC or RAdAC [27]. They rely on the session state space to examine all the packets and execute a more controlling access policy. This is called a *stateful inspection*. The states of each connection are stored in a datastore (e.g., database) for the duration of the session. They might include details such as the IP addresses, ports, the destinations, and the sequence numbers of the packets being transferred. It uses these stored states to decide what response to give to a requestor. In a certain sense, the datastore is the memory of the firewall policy. There are also software defined firewalls that are mainly stand-alone applications running in the background of a computer or on an access point to a local network. Hence, whether we are considering current hardware firewalls or software firewalls, we have a state space that encompasses the set of states governed by the access policies. The stateful control of access to resources is in use more and more to deal with growing sophistication in the attacks on networks. For example, we see increasingly that firewalls limit the number of embryonic connections to shield the network from Denial-of-Service (DoS) attacks. Or, for instance the ASA uses the per-client limits and the embryonic connection limit to trigger Transmission Control Protocol (TCP) Intercept, which protects inside systems from a DoS attack that is

perpetrated by flooding an interface with TCP SYN packets. These attack preventative activities require a memory and association with each packet that is examined at the firewall, which we refer to as the state space of the packet. Our work in this paper explores the use of the state space to reason on access control policies. One should look at policies as a special kind of program specification that should abide by the laws governing program specifications.

```
1  -A INPUT -s 156.17.49.0/24 -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 4000 -j ACCEPT
2
3  -A INPUT -s 156.17.49.0/24 -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 22 -j ACCEPT
4
5  -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
6
7  -A INPUT -p icmp -j ACCEPT
8
9  -A INPUT -p udp -m multiport --dports 5353 -j ACCEPT
10
11 -A INPUT -j REJECT --reject-with icmp-host-prohibited
```

Fig. 1. An Example of Firewall Policy

Figure 1 shows an example of an iptables[‡] firewall policy. We interpret the rule in Line 1 as the following command with a guard and an action. We use the notation $g \longrightarrow a$, where g is a guard and a is an action (as defined further in Definition 4).

$$\begin{aligned}&(\text{Direction} = \mathit{Input}) \wedge (\text{SourceIP} \in [156.17.49.0/24] \wedge (\text{Protocol} = \mathsf{TCP})\\ &\quad \wedge (\text{State} \in \{\mathsf{NEW}, \mathsf{RELATED}, \mathsf{ESTABLISHED}\}) \wedge (\text{DestinationPort} = 4000)\\ &\quad \longrightarrow \text{Action} = \mathsf{ACCEPT}\end{aligned}$$

We can similarly give the corresponding command to Line 3 in Figure 1. Also, we can directly combine the commands of Lines 1 and 3 into the following command, where the change from the previous one is only in the underlined condition. Therefore, a concrete policy can be interpreted as a set of commands or a single command obtained by combining in a coherent way all the commands as we did with Lines 1 and 3.

$$\begin{aligned}&(\text{Direction} = \mathit{Input}) \wedge (\text{SourceIP} \in [156.17.49.0/24] \wedge (\text{Protocol} = \mathsf{TCP})\\ &\quad \wedge (\text{State} \in \{\mathsf{NEW}, \mathsf{RELATED}, \mathsf{ESTABLISHED}\}) \wedge \underline{(\text{DestinationPort} \in \{4000, 22\})}\\ &\quad \longrightarrow \text{Action} = \mathsf{ACCEPT}\end{aligned}$$

In the following section, we present the background that relates the rules in an access policy to the mathematical concepts that allow us to reason on policies and on the strategies to deploy them on the firewalls of a network.

2.2 Mathematical Background

[‡] *iptables* is a command line utility for configuring Linux kernel firewall implemented within the Netfilter project

Product family algebra The paradigm of *product line* or *product family* in general has been transferred from hardware to software. Plainly, a *product family* is a set of products that share common hardware or software artefacts such as hardware components, requirements, architectural properties, middleware, or (in our case) security policies. A subfamily of a family A is a subset with elements sharing more features than the rest of the members of A. Sometimes, for practical reasons (i.e., managerial, or resource related), a specific software subfamily is called a *product line*.

One can think that when dealing with security, a policy can be a manifestation of the notion of product when one is reasoning on all the family of policies deployed all through a network. A feature is a conceptual characteristic that is visible to stakeholders (e.g., users, customers, developers, managers, etc.). In this paper, relevant stakeholders are security officers or any other organization actor who has a say on access control policies. Policies governing access to similar resources can be referred to as *policy family* or *policy product family*. We will base our theoritical results on PFA [9,10,11] that is briefly presented below.

Feature models, which are the means to give the mandatory, optional and alternative features within a domain, are used to represent families. They are widely used in product-line engineering to capture the commonality and variability of product families in terms of features. Using small feature models (in size of their graphs) can help to further guide distributing policies on firewalls or finding common rules among policies. However, the increasing complexity of network systems and the scale of the policies governing them, reveals that a large feature model cannot be understood and analyzed if they are treated as a monolithic entity. A similar situation is observed in the general use of software feature models. However, when we adopt an algebraic language to specify product family, this problem is avoided as a family is captured by an algebraic term and queries to feature models are carried through algebraic calculations. Algebraic approaches in general have the merit of being very suitable as lightweight formal methods with heavyweight automation [7]. Moreover, *point-free* reasoning, in the family of algebras based on variants of idempotent semirings such as PFA, can be formally linked with *point-wise* reasoning in concrete models, enabling us to switch back and forth between point-free abstract algebraic reasoning and point-wise concrete reasoning within a model [7]. In this paper, we use PFA not only to capture specific policies, but also to calculate the policy that should be assigned to each firewall of our network system.

Product family algebra (or briefly PFA) extends the mathematical notions of semiring to describe and manipulate product families. A *semiring* is an algebraic structure denoted by a quintuple $(S, +, \cdot, 0, 1)$, such that S is a set, $+$ and $\cdot$ are binary operations over S, and $0, 1 \in S$. The support set S is closed under $+$ and $\cdot$ operations. In particular, the binary operation $+$, called addition, is associative, and commutative, and has an identity element 0. The binary operation $\cdot$, called multiplication, is associative, and has an identity element 1. Multiplication left and right distributes over addition. Moreover, 0 is the annihilator element for multiplication. Furthermore, a commutative and idempotent semiring is a

semiring $(S, +, \cdot, 0, 1)$ such that multiplication is commutative, and addition $(+)$ is idempotent.

Definition 1 (Product Family Algebra (e.g., [11])). *A product family algebra is a commutative idempotent semiring $(S, +, \cdot, 0, 1)$, where*

a) *S corresponds to a set of product families;*
b) *$+$ is interpreted as the alternative choice between two product families;*
c) *$\cdot$ is interpreted as a mandatory composition of two product families;*
d) *0 corresponds to an empty product family;*
e) *1 corresponds to a product family consisting of only a pseudo-product which has no features.*

An optional feature f can be interpreted as an alternative choice between the feature f and 1. For example, let us consider policies p_1 that is assigned to a node N_1 and p_2 that is assigned to a node N_2. Nodes N_1 and N_2 are the only immediate successors on the graph representing a network to a node that we denote by N_0. The policies p_1 and p_2 share only the rules r_1 and r_2. However, p_1 has only one extra rule r_3. If we want, for example, to consider the policies that are employed starting from N_0, we represent them as a family $F \stackrel{\text{def}}{=} p_1 + p_2 = r_1 \cdot r_2 \cdot r_3 + r_1 \cdot r_2 = r_1 \cdot r_2 \cdot (r_3 + 1)$. The commonality of the members of the family F is the term $(r_1 \cdot r_2)$. If we look at product family algebras like the set-based or the the bag-based ones discussed in [11], we can formalize the problem of determining the commonality of two families as finding the Greatest Common Divisor (GCD), or to factor out the features common to all given products. We can use the classical Euclidean algorithm for finding the GCD, which is an advantage of using an algebraic approach. Solving the GCD is well known, easy and efficient, whereas finding commonalities using diagrams as used in several feature modelling approaches is more complex. We also have a divisibility relation among families that is given by $(a \mid b) \iff (\exists\, c \mid \cdot \quad b = a.c)$ [∓]. We say that two product families a and b are coprime iff $\mathsf{gcd}\,(a, b) = 1$.

A *requirement* relation over PFA is used to capture constraints in feature models. The *requirement* relation is defined using two other relations: *subfamily* and *refinement*. The subfamily relation indicates that, for two given product families a and b, a is a subfamily of b if and only if all of the products of a are also products of b. Formally, the subfamily relation $(\leq)$ is defined as $a \leq b \stackrel{\text{def}}{\iff} a + b = b$. For example, the above policy p_1 represents a subfamily of F that is given above, since we have $p_1 + F = p_1 + (p_1 + p_2) = p_1 + p_2 = F$. The refinement relation indicates that, for two given product families a and b, a

[∓] Throughout this paper, we adopt the uniform linear notation provided by Gries and Schneider in [8], as well as Dijkstra and Scholten in [4]. The general form of the notation is $(\star x \mid R \cdot P)$ where $\star$ is the quantifier, x is the dummy or quantified variable, R is predicate representing the range, and P is an expression representing the body of the quantification. An empty range is taken to mean true and we write $(\star x \mid \cdot P)$; in this case the range is over all values of variable x.

is a refinement of b if and only if every product in family a has at least all the features of some products in family b. Formally, the refinement relation ($\sqsubseteq$) is defined as $a \sqsubseteq b \stackrel{\text{def}}{\iff} (\exists\, c \mid \cdot \quad a \leq b \cdot c\,)$. In our example, we have $p_1 \sqsubseteq p_2$ as p_1 has all the rules of p_2 and more (the additional rule r_3). Also, we have $p_1 \sqsubseteq F$ as $p_1 \sqsubseteq F \iff (\exists\, c \mid \cdot \quad p_1 \leq F \cdot c\,) \iff (\exists\, c \mid \cdot \quad p_1 + F \cdot c = F \cdot c\,) \iff (\exists\, c \mid \cdot \quad r_1 \cdot r_2 \cdot r_3 + (r_1 \cdot r_2 \cdot (r_3 + 1)) \cdot c = (r_1 \cdot r_2 \cdot (r_3 + 1)) \cdot c\,)$, which is satisfied for $c = 1$ due to the idemptence of $+$.

An element $a \in S$ is said to be a *product* if it satisfies the following laws [9,11]:

$$(\forall\, b \mid b \in S \cdot b \leq a \implies (b = 0 \vee b = a)\,),$$

$$(\forall\, b, c \mid b, c \in S \cdot a \leq b + c \implies (a \leq b \vee a \leq c\,).$$

These laws define that a product cannot be divided using the choice operator $+$, or in other terms, it does not offer optional or alternative features. A *feature* can be defined by indivisibility w.r.t. multiplication rather than addition [9,11].

For elements a, b, c, d and a product p in PFA, the requirement relation ($\rightarrow$) is defined in a family-induction style [11] as:

$$a \xrightarrow{p} b \stackrel{\text{def}}{\Leftrightarrow} p \sqsubseteq a \Longrightarrow p \sqsubseteq b$$
$$a \xrightarrow{c+d} b \stackrel{\text{def}}{\Leftrightarrow} a \xrightarrow{c} b \wedge a \xrightarrow{d} b$$

The requirement relation is used to specify constraints on product families. For elements a, b and c, $a \xrightarrow{c} b$ can be read as "a requires b within c". The special case of a constraint $a \cdot b \xrightarrow{c} 0$ indicates that the composition of a and b generates an empty family. Such a constraint can be used to reflect the fact that not all feature compositions are possible or desirable in reality. For more details on the use of this mathematical framework to specify product families, we refer the reader to [9,10,11]. In our context, the constraints are used to express the will of security officers in the articulation of policies/rules applied to several access points. For example, we might need to state that if a user is denied access to resource x, then they must be denied (or allowed) access to resource y. These requirement rules, when taken into account, are very helpful for ensuring that the access policies capture the link among assess rules. Using PFA, a policy specifier can implement a set of policies and then constrain them using these requirement relations. Through calculations, the rules that breach these requirement constraints are eliminated. We say that a family f satisfies a constraint ($a \xrightarrow{q} b$), and we write ($(a \xrightarrow{q} b) \vdash f$), iff $(\forall\, p \mid p \leq f \wedge q \sqsubseteq p \cdot a \xrightarrow{p} b\,)$.

Commands, Guarded Commands, and if.fi-commands In this section, we present guarded commands as a proposed model for access control policies. We adopt a variant of Dijkstra's guarded command presented in [19,12]. Basically, a command is a transition relation from starting states to their possible successor

states. To guarantee the command does not have the possibility to lead to failure/abortion of a policy action, a command is modelled as a pair consisting of transition relation and a set of states for which no abortion is possible [21,20].

Definition 2 (e.g., [12]). *Consider a set Σ of states. A* command *over Σ is a pair (R, P) where $R \subseteq \Sigma \times \Sigma$ is a transition relation and P is a subset of Σ. The* restriction *of a transition relation $R \subseteq \Sigma \times \Sigma$ to a subset $Q \subseteq \Sigma$ is ${}_Q{\downarrow}R \stackrel{\text{def}}{=} R \cap (Q \times \Sigma)$.*

The set P is intended to characterize those states from which the command cannot lead to abortion. The command abort is the one that offers no transitions and does not exclude abortion of any state: $\mathsf{abort} \stackrel{\text{def}}{=} (\emptyset, \emptyset)$. It can be interpreted as the policy that does not involve any transitions on the state space or simply the absence of policy. Hence, since we have an absence of policy, there are no states that we trust to lead to normal termination of the policy command, which means we have an empty set P. There are other special commands that we will use in the remainder of the paper. For example, the command skip does not do anything: it leaves the state unchanged and cannot lead to abortion for any state: $\mathsf{skip} \stackrel{\text{def}}{=} (\mathbb{I}, \Sigma)$, where $\mathbb{I} \stackrel{\text{def}}{=} \{(s, s) \mid s \in \Sigma\}$ is the identity relation on states. The command fail does not offer any transition but guarantees that no state may lead to abortion: $\mathsf{fail} \stackrel{\text{def}}{=} (\emptyset, \Sigma)$. We now define the operators $[]$ of non-deterministic choice.

Definition 3 (e.g., [12]). *Let $C = (R, P)$ and $D = (S, Q)$ be commands. The command $C[]D$ is intended to behave as follows. For a starting state s, non-deterministically a transition under R or S is chosen (if there is any). Absence of aborting is guaranteed for s iff it can be guaranteed under both C and D, i.e., iff $s \in P \cap Q$. We define $[]$ as: $(R, P)[](S, Q) \stackrel{\text{def}}{=} (R \cup S, P \cap Q)$.*

The operation $[]$ is associative, commutative, and idempotent and fail is its neutral element. The reason for set union in the first and set intersection in the second is that if the choice of transitions gets greater, then the set of states for which no abortion is guaranteed gets smaller. We say that a command (R, P) is *feasible* when $P \subseteq dom(R)$.

Definition 4 (e.g., [12]). *Let (R, P) be a command and $Q \subseteq \Sigma$ be a set of states. Then the* guarded command $Q \longrightarrow (R, P)$ *(where Q is called the* guard*) is defined as $Q \longrightarrow (R, P) =_{df} ({}_Q{\downarrow}R, \overline{Q} \cup P)$, where $\overline{Q}$ is the complement of Q w.r.t. Σ.*

In a starting state s this command can lead to a transition only if s is in both Q and the domain of R (denoted by $\mathsf{dom}(R)$ and defined as $\mathsf{dom}(R) \stackrel{\text{def}}{=} \{s \in \Sigma \mid (\exists t \mid t \in \Sigma \cdot (s, t) \in R)\}$). Abortion is excluded if s is not in Q or P. Note that $Q \longrightarrow (R, P)$ is not feasible even if (R, P) is. Therefore, in [12], a way around this issue is proposed by defining the $\mathsf{if\_fi}$-statement.

Definition 5 (e.g., [12]). *Given a command (R, P), then the* $\mathsf{if\_fi}$*-statement is defined by* $\mathsf{if}\ (R, P)\ \mathsf{fi} \stackrel{\text{def}}{=} (R, P \cap \mathsf{dom}(R))$.

The reason the command is surrounded with if_fi is to transform it into a feasible command. This is used to define the semantic of the general construct of non-deterministic branching as follows. Given sets Q_i of states and commands (R_i, P_i), for $(1 \leq i \leq n)$, then

$$\begin{aligned}&\mathsf{if}\ Q_1 \longrightarrow (R_1, P_1)\ \mathsf{fi}[]\cdots[]Q_n \longrightarrow (R_n, P_n)\ \mathsf{fi} = \\ &\Big(\bigcup({}_{Q_i}{\downarrow}R_i),\ \ (\bigcup(Q_i \cap \mathsf{dom}(R))) \cap (\bigcap(\overline{Q_i} \cup P_i))\Big)\end{aligned}$$

We refer the reader to [12], from where the above definitions are taken, for more discussion on the if_fi construct and its mathematical properties.

In modelling access control rules, we use guarded commands. A guard ensures that the conditions implemented by a rule are satisfied before changing the state of the access system. A state change is done according to the transition relation of the command. Let $\mathrm{Dr}, S, P, \mathrm{St}, \mathrm{Ds}$, and A be respectively the sets of values of the directions (input, output), the source IP number, the protocols, the states, the destination ports, and the actions. We have $\Sigma = \mathrm{Dr} \times S \times P \times \mathrm{St} \times \mathrm{Ds} \times A$. Then, for example, the rule given on Page 5 and corresponding to Line 1 in Figure 1, that we call C_1 can be written as follows:

$$C_1 = [Q \longrightarrow (R, P)],\ \text{where}$$

$Q \subseteq \Sigma$ is the guard and defined as follows:

$\{(dr, s, p, st, ds, a) \mid\ (dr = \mathsf{Input})$
$\wedge\ (s \in [156.17.49.0 \cdots 156.17.49.24] \wedge (p = \mathsf{TCP}) \wedge (st \in \{\mathsf{NEW}, \mathsf{RELATED}, \mathsf{ESTABLISHED}\})$
$\wedge\ (ds = 4000)\}.$

The relation R can be defined in this case as

$$R = \{((dr, s, p, st, ds, a), (dr', s', p', st', ds', a'))\ \mid\ a' = \mathsf{ACCEPT}\},$$

and we take simply $P = \emptyset$; we are stating that without the guard, we cannot guarantee that the command avoids abortion. The guarded command C_1 corresponds to the guard $({}_{Q}{\downarrow}R, \overline{Q} \cup P) = (R \cap (Q \times \Sigma), \overline{Q} \cup \emptyset) = (R \cap (Q \times \Sigma), \overline{Q}) = (R \cap (Q \times \Sigma), \overline{Q})$. The second element of the tuple giving the guard (i.e., $\overline{Q}$) indicates that with the guard we are stating that all of the states outside of Q cannot lead to abortion.

We also take from the literature on guarded commands (e.g., [19,12]) the definition of the notion of *refinement* relation on commands. We say that (R, P) refines (S, Q) and we write $(R, P) \sqsubseteq (S, Q) \overset{\text{def}}{\Leftrightarrow} Q \subseteq P \wedge {}_{Q}{\downarrow}R \subseteq S$. This relation is reflexive, transitive, and not antisymmetric. The associated equivalence relation is given by $C \equiv D \overset{\text{def}}{\Leftrightarrow} C \sqsubseteq D \wedge D \sqsubseteq C$. In [12], the authors define equivalence of commands as $(R, P) \equiv (S, Q) \overset{\text{def}}{\Leftrightarrow} P = Q \wedge {}_{P}{\downarrow}R = {}_{P}{\downarrow}S$. We find also that the if_fi-construct is the "closest feasible refinement" of a command. We have $\mathsf{if}\ (R, P)\ \mathsf{fi}$ is the $\sqsubseteq$-least refinement of (R, P) that preserve the transition R. Then we find in [12] the following relation between the refinement relation and

non-deterministic choice: for commands C, D we have $C \sqsubseteq D \Leftrightarrow C [] D \equiv D$. Hence, two classes are related by $\sqsubseteq$ if their representatives are, which defines a partial order on equivalence classes of commands. We can imply from the above that the equivalence class of $C [] D$ is the least upper bound of the equivalence class of C and D w.r.t. $\sqsubseteq$. We can also define *greatest lower bound of commands* (R, P) and (S, Q) w.r.t. $\sqsubseteq$ as $(R, P) \sqcap (S, Q) = \Big((R \cap S) \cup (\overline{P}{\downarrow} S) \cup (\overline{Q}{\downarrow} R), \quad P \cup Q \Big)$. For two relations R and S, the meet of the feasible commands $(R, \mathsf{dom}(R))$ and $(S, \mathsf{dom}(S))$ is feasible iff $dom(R \cap S) = dom(R) \cap dom(S)$. In other terms, the meet of the feasible commands $(R, \mathsf{dom}(R))$ and $(S, \mathsf{dom}(S))$ is feasible iff R and S agree on the action to be carried on their common domain. It entails that for every state in the intersection of R and S we have to offer at least one transition. This allows for a common specification for the integration of R and S. This property is called *integrability*. Verifying the integrability of commands is a task that can me automated; it has been used before for the integration of requirement scenarios and has been automated using Prototype Verification System (PVS) [3,17]. In Section 4, we give an idea on the automation of the verification of the integrability of two commands in the paper's context. We note also that $\sqcap$ is commutative, associative, and has abort as its neutral element and fail as its absorbing element. Also, $[]$ and $\sqcap$ distribute over each other, which give the commands a distributive lattice structure. We refer the reader to [19,12], for further discussion on the *greatest lower bound of commands* (R, P) and (S, Q) w.r.t. $\sqsubseteq$.

3 Firewall Policies as Product Families

We showed the link between access control rules, guarded commands, and if_fi-statements. Using operations on commands $[]$ and $\sqcap$, we can define composite and quite complex commands. These cammands can be either simple access control rules or policies obtained by combining commands. Let G be the set of mutually integrable if_fi-statements. Let $\mathbb{P} \stackrel{\text{def}}{=} \mathscr{P}(G)$. The elements of $\mathbb{P}$ are called *Attribute Based Access Control Policies* (ABACP). For $A, B \in \mathbb{P}$, we define $A \sqcap_{\mathbb{P}} B \stackrel{\text{def}}{=} \{a \sqcap b \mid a \in A \wedge b \in B\}$. We can see that $a \sqcap_{\mathbb{P}} \{\mathsf{abort}\} = a$ as abort is neutral for the $\sqcap$ on commands. Now, we can state the following:

Proposition 1. *$\mathscr{F} = (\mathscr{P}(\mathbb{P}), \oplus, \odot, 0_{\mathscr{F}}, 1_{\mathscr{F}})$ is a product family algebra, where*

1. $(\forall A, B \mid A, B \in \mathscr{P}(\mathbb{P}) \cdot A \oplus B \stackrel{\text{def}}{=} A \cup B)$
2. $(\forall A, B \mid A, B \in \mathscr{P}(\mathbb{P}) \cdot A \odot B \stackrel{\text{def}}{=} \{a \sqcap_{\mathbb{P}} b \mid a \in A \wedge b \in B\})$
3. $0_{\mathscr{F}} \stackrel{\text{def}}{=} \emptyset$
4. $1_{\mathscr{F}} \stackrel{\text{def}}{=} \{ \{\mathsf{abort}\} \}$

The above proposition states that $\mathscr{F}$ is a model for PFA. $A \odot 0_{\mathscr{F}} = \{a \sqcap_{\mathbb{P}} b \mid a \in A \wedge b \in 0_{\mathscr{F}}\} = \{a \sqcap_{\mathbb{P}} b \mid a \in A \wedge b \in \emptyset\} = \{a \sqcap_{\mathbb{P}} b \mid a \in A \wedge \mathsf{false}\} = \{a \sqcap_{\mathbb{P}} b \mid \mathsf{false}\} = \emptyset = 0_{\mathscr{F}}$. Also, we have $A \odot 1_{\mathscr{F}} = \{a \sqcap_{\mathbb{P}} b \mid a \in A \wedge b \in 1_{\mathscr{F}}\} = \{a \sqcap_{\mathbb{P}} b \mid$

$a \in A \wedge b \in \{\ \{\mathsf{abort}\}\ \}\ \} = \{a \sqcap_{\mathbb{IP}} \{\mathsf{abort}\}\ \mid\ a \in A\} = \{a\ \mid\ a \in A\} = A$. Hence, $0_{\mathscr{F}}$ is the annihilator element for $\odot$ and $1_{\mathscr{F}}$ is the neutral for $\odot$. It is easy to see due to the properties of set union and the the operation $\sqcap$ on commands that $(\mathscr{P}(\mathbb{IP}), \oplus, \odot, 0_{\mathscr{F}}, 1_{\mathscr{F}})$ satisfies all the properties of an idempotent semiring and therefore it is a product family algebra. An element of $\mathscr{P}(\mathbb{IP})$ is called a *Family of Attribute Based Access Control Policy* and for brevity we say *family of policies*. On a product family, we a have a natural order that comes with the semiring structure that we denote for $\mathscr{F}$ by $\preceq_{\mathscr{F}}$. It is defined as $a \preceq_{\mathscr{F}} b \overset{\text{def}}{\Leftrightarrow} a \oplus b = b$. Hence, as discussed in Section 2.2, we can define a notion of *family refinement* of the elements of $\mathscr{F}$ as follows: $a \sqsubseteq_{\mathscr{F}} b \overset{\text{def}}{\Leftrightarrow} (\exists\, c\ \mid \cdot\quad a \preceq_{\mathscr{F}} b \odot c\,)$. For reasons of conciseness, we do not discuss the relationship between the command refinement to that of the family refinement. Obviously, they are linked.

We also, can instantiate the requirement relation defined in Section 2.2 in the structure $\mathscr{F}$ as it is a model of a product family algebra as stated in Proposition 1. For elements a, b, c, d and a product p in F, the requirement relation $(\rightarrow)$ is defined[†] in a family-induction style as:

$$a \xrightarrow{p} b \ \overset{\text{def}}{\Leftrightarrow}\ p \sqsubseteq_{\mathscr{F}} a \implies p \sqsubseteq_{\mathscr{F}} b$$
$$a \xrightarrow{c \oplus d} b \ \overset{\text{def}}{\Leftrightarrow}\ a \xrightarrow{c} b \wedge a \xrightarrow{d} b.$$

A relation $a \xrightarrow{p} b$ is called a Policy Requirement Constraint (PRC). It states that, within the family of policies p, if we satisfy the policies within family a, then we must satisfy the policies within family b. We usually use PRCs to express global network access policies. When we want to articulate the constraint that, in family of policies p, we should not satisfy the policies in family a we write $a \xrightarrow{p} 0_{\mathscr{F}}$. In other terms, we are stating that no policies in family P should refine any policy in family a.

3.1 Defense in Depth Strategy and its Usage

When we consider a resource network that has an access entry r allowing its access from the outside world, we can represent it as a rooted connected directed acyclic graph. The leafs of the graph represent the resources to be accessed. The remaining vertices would be internal access nodes that execute policies. Figure 2 shows the graph model of a network that has a root r, leafs v_6 to v_{10}, and internal access points v_1 to v_5. All the vertices can execute policies. The edges represent access traffic links between access points. For example, in Figure 2, the edge (v_1, v_4) indicates the access connection from access node v_1 to access node v_4.

We might have networks with n entry points. In this case, we model it with n rooted connected directed acyclic graphs that each has one of the entry points as its root. The formal treatment presented below would need to be repeated to each rooted connected directed acyclic graph. Then, each of the network access points, would enforce a family of policies that is the sum of all the families of policies associated to it and obtained from each of the rooted graphs.

[†] As it is a simple instantiation in a model of PFA of that of Section 2.2, we use the same notation.

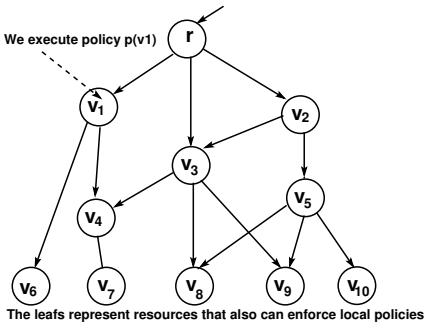

Fig. 2. A Resource Network as a Rooted Connected Directed Acyclic Graph

Let $G \stackrel{\text{def}}{=} (V, E, r)$ be a rooted connected directed acyclic graph that represents a resource network, where:

- V is the set of vertices and it represents the set of access control points that enforce access policies;
- E is a set of ordered pairs of vertices that represent the link between access control points;
- r is the root of the graph and it represents the access point between the network and the external word.

From now on, we call G a *network of access control points*.

Definition 6 (Defence in Depth Law (DDL)). *Let $G \stackrel{\text{def}}{=} (V, E, r)$ be a network of access control points. We denote by $p(v)$ the family of attribute based access control policies enforced by vertex v in G. The network G employs a DD strategy if* $p(r) \neq 0_{\mathscr{F}} \ \wedge \ (\forall\, a, b \mid (a, b) \in E \cdot p(b) \sqsubseteq_{\mathscr{F}} p(a))$

Obviously, if a node v satisfying $(r, v) \in E$ has $p(v) = 1_{\mathscr{F}}$, then the root will have $p(r)$ satisfying $1_{\mathscr{F}} \leq p(r) \iff 1_{\mathscr{F}} \oplus p(r) = p(r)$. It is because $1_{\mathscr{F}}$ can refine $1 + c$ for some c, or refine $0_{\mathscr{F}}$. The $0_{\mathscr{F}}$ is not allowed as it is the "impossible" family of policies. The family $1_{\mathscr{F}}$ contains only one policy with one rule given by the command $\mathsf{abort} \stackrel{\text{def}}{=} (\emptyset, \emptyset)$, which offers no transitions (no change of state) and does not exclude abortion of any state. The second condition ensures that every policy at a level higher than the root (we assume the root to be at level 0; the lowest) needs to be at least as restrictive, if not more, than the one above it. This fact is articulated explicitly in Proposition 2(a).

Definition 6 does not prevent trivial instances in which all access control happens at the leaves and all other nodes accept all traffic. Practically this situation could happen when we adopt for instance the approach given in Proposition 4 for co-prime policies executed at the leafs (resources) and without global constraints; no way to have a common restrictive rule that can be applied at their ancestor nodes. In this case, we are forced to let the resources enforce the rules

and allow each to accept the traffic only destined to them. Otherwise, any control at a node upstream would block access to some resources.

One can think of a more strict form of DD than that of Definition 6 by strengthening the condition to prevent trivial instances in which all access control happens at the leaves and all other nodes accept all traffic. It would simply require to change the refinement relationship between $p(a)$ and $p(b)$ in the condition of Definition 6 to a strict refinement as follows:

$$p(r) \neq 0_{\mathscr{F}} \ \wedge \ (\forall\, a, b \ \mid \ (a,b) \in E \ \cdot \ p(b) \sqsubset_{\mathscr{F}} p(a)\,),$$

where $p(b) \sqsubset_{\mathscr{F}} p(a) \ \Longleftrightarrow \ (p(b) \sqsubseteq_{\mathscr{F}} p(a) \ \wedge \ p(a) \neq p(b))$. In the rest of this paper, we adopt the weak form of DD that is given in Definition 6 for the simplicity that it provides to the treatment of DD.

Proposition 2. *Let G be a network of access control points that employ a DD strategy. Let $P = \langle v_1, v_2, ..., v_m \rangle$ be a path of P. We have*

a) $(\forall\, i \ \mid \ 1 \leq i \leq m \ \cdot \ p(v_m) \sqsubseteq_{\mathscr{F}} p(v_i)\,)$

b) $(\forall\, v \ \mid \ v \in E \ \cdot \ p(v) \sqsubseteq_{\mathscr{F}} p(r)\,)$

Proof. The proof for item a uses the reflexivity and transitivity of $\sqsubseteq_{\mathscr{F}}$ and some basic quantifier rewriting rules. While the proof for item (b) is done by induction on $Q(m) \stackrel{\text{def}}{\Leftrightarrow} (\forall\, i \ \mid \ 1 \leq i \leq m \ \cdot \ p(v_i) \sqsubseteq_{\mathscr{F}} p(v_1)\,)$. The detailed proof is given in the Appendix.

The result 2(a) states that whatever path the access takes in a network that implements a DD strategy, it will be faced by more and more restrictive (in a weak sense) families of policies. The result 2(b) states that any family of policies at any of the network nodes is at least as restrictive as that of the root.

3.2 Generating Lower Level Policies from Higher Level Ones

In a network of access control points $G \stackrel{\text{def}}{=} (V, E, r)$, we assign level 0 to r. We say that r has the lowest level in G. Let v_i be a vertex having level n, then a vertex v_j such that $(v_i, v_j) \in E$ will have the level $n+1$. A vertex might have more than one level as it might be reached by several paths of different lengths. Only when G is a tree, the vertices have unique levels.

Proposition 3. *Let $G \stackrel{\text{def}}{=} (V, E, r)$ be a network of access control points. Let T be a directed spanning tree of G rooted at r and having a set L of leaves. For every $l \in L$, we are given $p(l)$. If we have*

$$p(v) \stackrel{\text{def}}{=} (\oplus\, v_i \ \mid \ (v, v_i) \in E \wedge p(v_i) \neq 1_{\mathscr{F}} \ \cdot \ p(v_i)\,)$$

for every $v \in V$ that is an ancestor of an $l \in L$, then G employs a DD strategy.

Proof. Since for every $v \in E$, we have $p(v)$ is constructed using the operator $\oplus$ of all the families of policies that are enforced at nodes that come after node v. Therefore, each of these policies refines $p(v)$ which satisfies the condition in Definition 6.

This proposition enables assigning policies starting from the highest (level) vertices in the network (i.e., the resource). We start by manually assigning the access policies to the resources (leaves in the tree). Then the policies of the lower nodes are generated according to the scheme proposed by Proposition 3.

In the case where G has several spanning trees $T_1 \cdots T_j$, for $j \leq |V|^{(|V|-2)}$ (as for a complete graph with n vertices, Cayley's formula gives the number of spanning trees as n^{n-2}), then a vertex $v \in G$ belongs to each of the spanning trees has a family of policies $p(v) = (\oplus i \mid 1 \leq i \leq j \cdot p_i(v))$, where $p_i(v)$ is the family of policy for vertex v obtained according to Proposition 3 using the spanning tree T_i for $1 \leq i \leq j$. In this context, the family approach to deal with policies where many paths from the root can lead to one access control point is very convenient; we have a family of policies that apply not only to one tree.

We suggest in the next proposition another deployment scheme of families of policies.

Let $G \stackrel{\text{def}}{=} (V, E, r)$ be a network of access control points. Let T be a directed spanning tree (DST) of G rooted at r and having a set L of leaves. For every $l \in L$, we are given $p(l)$.

Proposition 4. *If we have* $p(v) = (\,\mathsf{gcd}\ v_i \mid (v, v_i) \in E \wedge p(v_i) \neq 1_{\mathscr{F}} \cdot p(v_i)\,)$ *for every* $v \in V$ *that is an ancestor of an* $l \in L$*, then* G *employs a DD strategy.*

Proof. The proof uses the fact that in a product family algebra, $a \cdot c \sqsubseteq a$. This is true in our model of product family (i.e., $a \odot c \sqsubseteq_{\mathscr{F}} a$). At the node v, we deploy the family of policies that is given by the commonality of the policies at v_i.

If one of the v_i for $(v, v_i) \in E$ is coprime to one of the others v_j at the same level and that are related to v, then $p(v) = 1_{\mathscr{F}}$. Two families are coprime indicates that they do not have policies/rules that are shared by the two of them.

The following proposition is about the preservation of the defense in depth when we apply PRCs.

Proposition 5. *Given a network of access control points* G *that employs a DD strategy, where each node* v *has a family of policies* $p(v)$ *assigned to it. Let* C *be a given set of PRCs. The following scheme gives a network that employs a DD strategy.*

For every $v \in V$ *that is an ancestor of an* $l \in L$*, we assign a family of policies* $p'(v)$ *such that*

1. $p'(v) \leq p(v)$*, and*
2. $(\forall\, c, v, w \mid c \in C \wedge (v, w) \in E \cdot (p(w) \leq p(v)) \wedge (c \vdash p'(v)) \wedge (c \vdash p'(w)))$.

Proof. Since $p'(v)$ is a subfamily of $p(v)$ and the refinement between a node and its successor on the tree L is reduced to the subfamily relationship. Therefore, applying the constraints preserves the refinement needed for the DD strategy. Without the condition of $p(w) \leq p(v)$ as given above, there is no guarantee that the refinement is preserved by applying the PRCs.

4 Automation of the Management of Policies and the Verification of their Integrability

In this section, due to space limitation, we simply point to the main components of our prototype tool and the technology used to automate the results proposed in this paper. Our prototype tool includes two major elements: Analysis element (*Analyzer*) and broker element (*Broker*).

The *Analyzer* is responsible for all the calculations needed to ensure the integrability of policies and for assigning policies (according to one of the schemata given in Propositions 3, 4, or 5) to each access control node based on the given policies assigned to the resources.

The *Broker* has the responsibility to keep track of the policies at each node and to transmit newly calculated policies to their corresponding nodes. Each node subscribes with the *Broker*, notifies it of any change to its situation. Then, the *Broker*, with the help of the *Analyzer*, decides on the appropriate policy for each node, and transmits them to their destinations. The design of the *Broker* is based on the *observer* pattern, which is a software design pattern in which an object maintains a list of its observers (in our case the nodes to be assigned policies) and notifies them automatically of any state changes (policies changes), usually by calling one of their methods. Hence, the *Broker* construction is a straight forward application of *observer* design pattern.

The policies for each of the network resources are automatically translated into tabular expressions commonly known as Parnas' Tables (e.g., [22,16]). A tabular expression can be encoded using a markup language. In our prototype, we use a language that has been introduced in [17]. The *Analyzer* of our prototype tool uses PVS to perform the verification and calculations needed whether for verifying policies or for determining appropriate policies for each node. It has been demonstrated in [23] that PVS is an appropriate theorem prover for carrying calculations using a formalism similar to the one we are using in this paper. The *Analyzer* is a modified version of the tool SCENATOR [17] developed for the verification of requirements scenarios. The formalism used in SCENATOR is similar to the one we are using for the analysis of security policies. The main addition to SCENATOR is the development of modules to automatically calculate the GCD of a family of policies. The *Analyzer* is implemented using C, Tcl\TK, and runs on Unix/Linux platform. It uses PVS in batch mode. If two policies are not integrable, it highlights in their corresponding tables, the cells that are inconsistent. Also, when given families of policies, it performs calculations such as the GCD of the members of a family or calculates the operations defined on families of policies. The approach for generating conjectures for PVS to prove

(in batch mode) and how the results are interpreted are thoroughly discussed in [17]. We simply reused the existing tool SCENATOR with the few additions described above.

Our prototype is only a proof of concept for the automation of a dynamic access control. Issues related to time-length of control cycles and the observability of changes to policies or to the states of the resources need to be considered with more care and precision for an efficient dynamic access control solution. Moreover, from a design perspective, the question on how to prevent the *Broker* from becoming a target of attacks needs to be addressed. These are issues that require further investigations.

5 Discussion and Future Work

We think that a family approach is appropriate to reason on the access policies of a network for the following motives: (1) We distinguish between the actual specific implementation of the policies and the family of policies coming from several viewpoints that gave that implementation. A family of policies can give other implementations such as the concurrent version that most current firewall technologies do not support. However, the requirements of today's technology demands for the enhancement of resource access performance; especially in this era of *Internet of Things* where a large number of devices can create resource access contention. (2) Using a family approach keeps the separation of concerns in the considered family of policies. Any change to a policy usually concerns one view point (coming from one security stakeholder) and therefore it is easy to locate and carry the change. Then, in a systematic way, we generate the actual implementation of the family as a sequentially executed list of rules or as a set of rules that are safe to be executed in any order (which our proposed model allows). Having correct methods for automatically and dynamically verifying these changes and reconfiguring firewalls would be a step towards a dynamic approach to a system's access control. (3) Adopting a family approach to reason on access policies, as we presented, enables us to not rely on a person to articulate the policy into a sequence of rules where an alteration in the order of execution of two rules can threaten the security of our resources. A systematic way should be adopted to generate the actual policy so that its function is independent of the order of execution of its rules.

Another context where the usage of a family approach is beneficial is when reasoning on the overall security of a network of resources. Let us consider, in a network, a node N under which we are running n virtual machines, where each machine has its own access policy. Abstractly, when we want to reason on the whole network security, we can consider that at node N, we are executing a family of policies where its members are each of the n virtual machine's access policy. We can use this abstract approach to go up layer by layer until we have constructed the family of policies under the control of the root of the network. Moreover, in Section 3, we proposed other usages of families of access policies, such as defining the defense in depth strategy and presenting several of its im-

plementations. Also, when we consider a node that can be accessed from several paths from the root, it has several policies that can be executed depending on the path taken by the access request. It is the case when we have several spanning trees in the network (case discussed in Section 3.1). In this case, a family approach is more intuitive in reasoning on security policies.

Articulating and implementing access control security policies is no different than other similar activities related to enterprise security policies. When an organization is faced with several challenging priorities, the business rational drives prioritizing resources and dedication to each activity. The challenge is that resource access control policies can become low-priority and their maintenance and management will be assigned to technical staff that do their best as they see fit. Automating the process of verifying policies as they are introduced or as they are amended will ensure that, even with few resources, the security system behind the network can take care of itself in configuring and implementing policies. However, to ensure sound automation, a formal background is needed to base on it the detection, recovery, and prevention mechanisms. The ideas and the schemata for assigning security policies presented in this paper give the background for this automation. Indeed, we developed an access control policy software that does the verification of the integrability of policies as discussed on Page 11 and assigns policies to nodes according to Propositions 3–5.

We find in [13] a product family approach to relate the security policies to the security functionalities. A security policy is enforced through the deployment of certain security functionalities within the application. Then, to handle the issue of frequent changes in security policy requirements they adopt an aspect oriented approach. This issue is also present in articulation and deploying access control policies. A means to quickly deal with changes to the rules is a must. Sometimes, when a security flaw is discovered, we are required to replace some conditions by others that address the problem and apply that change to all the policies. An aspect oriented approach would be appropriate for quickly propagating the correction to all the policies. In our case, we build our work on PFA. An extension of PFA, which is Aspect-Oriented Product Family Algebra (AO-PFA) [30,31], has an aspect oriented language. Moreover, recently, Zhang et al. [29] proved that its weaving process is convergent, leads to unambiguous weaving results, and that its rewriting system is terminating and confluent. As our formalism is based on PFA, we will be able to easily handle the issues of weaving policy changes to their corresponding policies and that at the right join points. However, the need for assessing the affect of these changes to all the access control nodes and the affect on the DD strategy remains to be investigated.

Our proposed approach requires a quite heavy calculational effort as well as some of its decision functions are, in general, undecidable. For example, determining whether two commands are integrable is undecidable in general. When we use SCENATOR [17], if there is an undecidability problem, the cells, in the used tabular expressions, that give rise to undecidability will be marked and the security analysis will be considering it and making the appropriate decision. In practice, these commands that their integrability verification is undecidable, can

be amended into decidable cases by, for example, restricting the state space (a discussion on this issue can be found in [3]). Using PVS of the required logical calculations is straight forward and can scale to handle large network system.

Some might argue that this approach is state based and with a large network we might observe a state space explosion. This point has some merit, however we should keep in mind the following: (1) The state space does not increase with each resource that is added. We alway consider a quite stable set of attributes of the network such as the source, the destination, the user, the protocol, etc. The dimensions of our space is some what stable. (2) We can divide a complex and quite large network of resources into subnetworks and we assign a policy *Broker* (as described above) for each subnetwork. However, the constraints of one subnetwork on the other can be seen as global constraints and be handled as prescribed by Proposition 5. We are opting for a centralized approach to assign policies. In [28], Google's tech lead for networking and others argue that, for traffic control, a central perspective allows to make better decisions.

6 Conclusion

As far as we know, in the context of access control policies within a network, the paper formally captures for the first time the widely intuitively discussed Defense in Depth strategy. It allowed us to formally assess whether a network, with a given topology and a set of policies distributed on it, satisfies the DD strategy or not. We point to a stronger version of DD strategy that might not possible for any given set of policies. Moreover, we can articulate several sound schemata for assigning policies such that the configuration of policies on a network satisfy the DD strategy. The schemata presented in Propositions 3, 4, and 5 constitute an effort to automatically distribute security policies that satisfy the DD strategy. They can be used to allow a dynamic reconfiguration of firewalls policies each time there is a change to the access policy of a resource (Propositions 3, 4) or moreover when there is a change in the set of overall access-constraints put on the network (Proposition 5). This dynamic aspect of reconfiguration of firewall policies after each modification creates a kind of mobile defense. It makes predicting a policy that is executed on a firewall more difficult. This hinders mounting attacks on the system or at least makes them more challenging due to the mobility of the rules between firewalls (e.g., due to their change each time a resource is temporally unavailable, or because a resource reached its load capacity). Moreover, the access to a resource is granted by all firewalls on the path to the resource. This presents a *separation of duties* that is a key concept of internal controls. It is achieved by disseminating the tasks and associated privileges for a specific security process among multiple firewalls on the path to a resource, so that compromising a single node does not, in general, compromise the network.

More work needs to focus on articulating more efficient schemata that fit some given criteria. In this paper, we examined involving global access policies (i.e., PRCs). However, one can think about other performance related criteria that can affect the distribution of policies.

Appendix: Detailed Proof of Proposition 2

Proof. a) $(\forall\, i \mid 1 \leq i \leq m \cdot p(v_m) \sqsubseteq_{\mathscr{F}} p(v_i))$

$\Longleftrightarrow$ $\langle\ 1 \leq i \leq m \iff 1 \leq i \leq m-1 \vee i = m\ \rangle$

$(\forall\, i \mid 1 \leq i \leq m-1 \vee i = m \cdot p(v_m) \sqsubseteq_{\mathscr{F}} p(v_i))$

$\Longleftrightarrow$ $\langle$ Range Split and One Point Axiom, and Reflexivity of $\sqsubseteq_{\mathscr{F}}$ $\rangle$

$(\forall\, i \mid 1 \leq i \leq m-1 \cdot p(v_m) \sqsubseteq_{\mathscr{F}} p(v_i)) \wedge \mathsf{true}$

$\Longleftrightarrow$ $\langle$ Identity of $\wedge$ $\rangle$

$(\forall\, i \mid 1 \leq i \leq m-1 \cdot p(v_m) \sqsubseteq_{\mathscr{F}} p(v_i))$

$\Longleftrightarrow$ $\langle\ 1 \leq i \leq m-1 \iff 1 \leq i \leq m-2 \vee i = m-1\ \rangle$

$(\forall\, i \mid 1 \leq i \leq m-2 \vee i = m-1 \cdot p(v_m) \sqsubseteq_{\mathscr{F}} p(v_i))$

$\Longleftrightarrow$ $\langle$ Range Split and One Point Axiom $\rangle$

$(\forall\, i \mid 1 \leq i \leq m-2 \cdot p(v_m) \sqsubseteq_{\mathscr{F}} p(v_i)) \wedge p(v_m) \sqsubseteq_{\mathscr{F}} p(v_{m-1})$

$(\forall\, i \mid 1 \leq i \leq m-2 \cdot p(v_m) \sqsubseteq_{\mathscr{F}} p(v_i))$

$\Longleftrightarrow$ $\langle\ 1 \leq i \leq m-2 \iff 1 \leq i \leq m-3 \vee i = m-2\ \rangle$

$(\forall\, i \mid 1 \leq i \leq m-3 \vee i = m-2 \cdot p(v_m) \sqsubseteq_{\mathscr{F}} p(v_i))$

$\Longleftrightarrow$ $\langle$ Range Split and One Point Axiom $\rangle$

$(\forall\, i \mid 1 \leq i \leq m-3 \cdot p(v_m) \sqsubseteq_{\mathscr{F}} p(v_i)) \wedge p(v_m) \sqsubseteq_{\mathscr{F}} p(v_{m-2})$

$\Longleftrightarrow$ $\langle$ Since $(v_{m-2}, v_{m-1}) \in E \implies p(v_{m-1}) \sqsubseteq_{\mathscr{F}} p(v_{m-2})$ and transitivity of $\sqsubseteq_{\mathscr{F}}$ $\rangle$

$(\forall\, i \mid 1 \leq i \leq m-3 \cdot p(v_m) \sqsubseteq_{\mathscr{F}} p(v_i)) \wedge \mathsf{true}$

$\Longleftrightarrow$ $\langle$ Identity of $\wedge$ $\rangle$

$(\forall\, i \mid 1 \leq i \leq m-3 \cdot p(v_m) \sqsubseteq_{\mathscr{F}} p(v_i))$

$\Longleftrightarrow$ $\langle$ Range Split several times and transitivity of $\sqsubseteq_{\mathscr{F}}$ $\rangle$

true

b) Let $Q(m) \stackrel{\text{def}}{\Leftrightarrow} (\forall\, i \mid 1 \leq i \leq m \cdot p(v_i) \sqsubseteq_{\mathscr{F}} p(v_1))$, for some $m \in \mathbb{N}$.

Base Case: $Q(1) \stackrel{\text{def}}{\Leftrightarrow} (\forall\, i \mid 1 \leq i \leq 1 \cdot p(v_i) \sqsubseteq_{\mathscr{F}} p(v_1))$, which is obviously true due to the *One Point Axiom* and the reflexivity of $\sqsubseteq_{\mathscr{F}}$.

Inductive Step: For arbitrary $m \geq 1$, we prove $Q(m+1)$ using the hypotheses ($Q(m)$ is true) and (G employs a DD strategy).

$(\forall\, i \mid 1 \leq i \leq m+1 \cdot p(v_i) \sqsubseteq_{\mathscr{F}} p(v_1))$

$\Longleftrightarrow$ $\langle\ 1 \leq i \leq m+1 \iff 1 \leq i \leq m \vee i = m+1\ \rangle$

$(\forall\, i \mid 1 \leq i \leq m \vee i = m+1 \cdot p(v_i) \sqsubseteq_{\mathscr{F}} p(v_1))$

$\Longleftrightarrow$ $\langle$ Range Split and One Point Axiom $\rangle$

$(\forall\, i \mid 1 \leq i \leq m \cdot p(v_i) \sqsubseteq_{\mathscr{F}} p(v_1)) \wedge p(v_{m+1}) \sqsubseteq_{\mathscr{F}} p(v_1)$

$\Longleftrightarrow$ $\langle$ From the hypothesis $Q(m)$ is true $\rangle$

$\mathsf{true} \wedge p(v_{m+1}) \sqsubseteq_{\mathscr{F}} p(v_1)$

$\Longleftrightarrow$ $\langle$ From (a), and Idompotency of $\wedge$ $\rangle$

true

References

1. Burns, J., Cheng, A., Gurung, P., Rajagopalan, S., Rao, P., Rosenbluth, D., Surendran, A.V., Martin, D.M.: Automatic management of network security policy. In: DARPA Information Survivability Conference & Exposition II (DISCEX '01), Volume 2. pp. 12 – 26. DARPA in cooperation with the IEEE Computer Society's Technical Committee on Security and Privacy, IEEE, Anaheim, CA (12 June – 14 June 2001) 2
2. Cheng, P.C., Rohatgi, P., Keser, C., Karger, P., Wagner, G., Reninger, A.: Fuzzy multi-level security: An experiment on quantified risk-adaptive access control. In: IEEE Symposium on Security and Privacy. pp. 222–230 (May 2007) 4
3. Desharnais, J., Frappier, M., Khedri, R., Mili, A.: Integration of sequential scenarios. IEEE Transactions on Software Engineering 24(9), 695 – 708 (September 1998) 11, 19
4. Dijkstra, E., Scholten, C.: Predicate Calculus and Program Semantics. Springer-Verlag New York, Inc., New York, NY, USA (1990) 7
5. Ferraiolo, D., Kuhn, R.: Role-based access control. In: 15th NIST-NCSC National Computer Security Conference. pp. 554–563 (1992) 4
6. Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed nist standard for role-based access control. ACM Trans. Inf. Syst. Secur. 4(3), 224–274 (August 2001) 4
7. Foster, S., Struth, G., Weber, T.: Automated engineering of relational and algebraic methods in isabelle/HOL. In: de Swart, H. (ed.) Relational and Algebraic Methods in Computer Science: 12th International Conference, RAMICS 2011 Rotterdam, The Netherlands, May/June 2011 Proceedings. Lecture Notes in Computer Science, vol. 6663, pp. 52–67. Springer-Verlag Berlin Heidelberg (2011) 6
8. Gries, D., Schenider, F.: A Logical Approach to Discrete Math. Springer Texts And Monographs In Computer Science, Springer-Verlag, New York (1993) 7
9. Höfner, P., Khedri, R., Möller, B.: Feature algebra. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006: Formal Methods. Lecture Notes in Computer Science series, vol. 4085, pp. 300 – 315. Springer, 14th International Symposium on Formal Methods, McMaster University, Hamilton, Ontario, Canada (August 21 – 27 2006) 6, 8
10. Höfner, P., Khedri, R., Möller, B.: Algebraic view reconciliation. In: 6th IEEE International Conferences on Software Engineering and Formal Methods. pp. 85 – 94. Cape Town, South Africa (November 10 – 14, 2008) 6, 8
11. Höfner, P., Khedri, R., Möller, B.: An algebra of product families. Software & Systems Modeling 10(2), 161–182 (2011) 6, 7, 8
12. Höfner, P., Khedri, R., Möller, B.: Supplementing product families with behaviour. International Journal of Software and Informatics pp. 245–266 (2011) 8, 9, 10, 11
13. Horcas, J.M., Pinto, M., Fuentes, L.: Closing the gap between the specification and enforcement of security policies. In: Eleftherakis, G., Hinchey, M., Holcombe, M. (eds.) Trust, Privacy, and Security in Digital Business, Lecture Notes in Computer Science, vol. 8647, pp. 106 – 118. Springer International Publishing (2014) 18
14. Hu, C.T., Ferraiolo, D.F., Kuhn, D.R., Schnitzer, A., Sandlin, K., Miller, R., Scarfone, K.: Guide to attribute based access control (abac) definition and considerations (January 2014) 4
15. Hu, V., Kuhn, D., Ferraiolo, D., Voas, J.: Attribute-based access control. Computer 48(2), 85–88 (2015) 4

16. Janicki, R., Khedri, R.: On a formal semantics of tabular expressions. Science of Computer Programming 39(1-2), 189–213 (March 2001) 16
17. Khedri, R., Wu, R., Sanga, B.: SCENATOR: A prototype tool for requirements inconsistency detection. In: Wang, F., Lee, I. (eds.) Proceedings of the 1st International Workshop on Automated Technology for Verification and Analysis. pp. 75 – 86. National Taiwan University, Taiwan, Republic of China (December 10 – 13 2003) 11, 16, 17, 18
18. Lippmann, R., Ingols, K., Scott, C., Piwowarski, K., Kratkiewicz, K., Artz, M., Cunningham, R.: Validating and restoring defense in depth using attack graphs. In: MILCOM 2006 - 2006 IEEE Military Communications conference. pp. 1 – 10 (Oct 2006) 3
19. Möller, B., Struth, G.: wp is wlp. In: MacCaull, W., Winter, M., Düntsch, I. (eds.) Relational Methods in Computer Science, Lecture Notes in Computer Science, vol. 3929, pp. 200–211. Springer Berlin Heidelberg (2006) 8, 10, 11
20. Parnas, D.L.: Precise description and specification of software. In: Software Fundamentals. Addison-Wesley (1997) 9
21. Parnas, D.L.: A generalized control structure and its formal definition. Commun. ACM 26(8), 572–581 (Aug 1983) 9
22. Parnas, D.L.: Tabular representation of relations. CRL Report 260, Communications Research Laboratory, Faculty of Engineering, McMaster University, Hamilton, Ontario, Canada (October 1992) 16
23. Rushby, J., Srivas, M.: Using PVS to prove some theorems of David Parnas. In: Joyce, J.J., Seger, C.J.H. (eds.) Higher Order Logic Theorem Proving and its Applications (6th International Workshop, HUG '93). Lecture Notes in Computer Science, vol. 780, pp. 163–173. Springer-Verlag, Vancouver, Canada (aug 1993) 16
24. Samarati, P., Vimercati, S.D.C.d.: Access control: Policies, models, and mechanisms. In: Revised Versions of Lectures Given During the IFIP WG 1.7 International School on Foundations of Security Analysis and Design on Foundations of Security Analysis and Design: Tutorial Lectures. FOSAD '00 (2001) 1
25. Sandhu, R., Ferraiolo, D., Kuhn, R.: The nist model for role-based access control: Towards a unified standard. In: Proceedings of the Fifth ACM Workshop on Role-based Access Control. pp. 47–63. RBAC '00, ACM, New York, NY, USA (2000) 4
26. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. Computer 29(2), 38–47 (1996) 4
27. Scarfone, K., Hoffman, P.: Guidelines on firewalls and firewall policy. Tech. rep., National Institute of Standards and Technology (NIST) (2009) 3, 4
28. Vahdat, A., Clark, D., Rexford, J.: A purpose-built global network: Google's move to SDN (a discussion with amin vahdat, david clark, and jennifer rexford). Commun. ACM 59(3), 46–54 (2016), http://doi.acm.org/10.1145/2814326 2, 19
29. Zhang, Q., Khedri, R.: On the weaving process of aspect-oriented product family algebra. Journal of Logical and Algebraic Methods in Programming 85(1, Part 2), 146 – 172 (January 2016), http://dx.doi.org/10.1016/j.jlamp.2015.08.004, special Issue on Formal Methods for Software Product Line Engineering 18
30. Zhang, Q., Khedri, R., Jaskolka, J.: Verification of aspectual composition in feature-modeling. In: Eleftherakis, G., Hinchey, M., Holcombe, M. (eds.) Software Engineering and Formal Methods, Lecture Notes in Computer Science, vol. 7504, pp. 109 – 125. Springer Berlin / Heidelberg (2012) 18
31. Zhang, Q., Khedri, R., Jaskolka, J.: An aspect-oriented language for feature-modeling. Journal of Ambient Intelligence and Humanized Computing 5, 343 – 356 (2014) 18