MIRAGE: a management tool for the analysis and deployment of network security policies

Joaquin Garcia Alfaro, Frédéric Cuppens, Nora Cuppens-Boulahia, Stere Preda

MIRAGE: A Management Tool for the Analysis and Deployment of Network Security Policies

J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia, and S. Preda

Abstract We present the core functionality of MIRAGE, a management tool for the analysis and deployment of configuration policies over network security components, such as firewalls, intrusion detection systems, and VPN routers. We review the two main functionalities embedded in our current prototype: (1) a bottom-up analysis of already deployed network security configurations and (2) a top-down refinement of global policies into network security component configurations. In both cases, MIRAGE provides intra-component analysis to detect inconsistencies in single component deployments; and inter-component analysis, to detect multi-component deployments which are not consistent. MIRAGE also manages the description of the security architecture topology, to guarantee the proper execution of all the processes.

1 Introduction

Despite the advances in the field of network security technologies, such as filtering of traffic, use of encrypted communications, and deployment of authentication mechanisms, there may always be errors or flaws that can be exploited by unauthorized parties. The use of firewalls, NIDSs (network intrusion detection systems), and VPN (Virtual Private Network) routers, is still the dominant method to survey and guarantee the security policy in current corporate networks. The configuration of these components is based on the distribution of security rules that state what is permitted and what is prohibited in a system during normal operations. This configuration must be consistent, addressing the same decisions under equivalent conditions, and not repeating the same actions more than once. Otherwise, the existence of anomalies in their configuration rules may lead to weak security policies (potentially easy to be evaded by unauthorized parties). The update of the component configurations can also introduce new anomalies. There is, therefore, a clear need of support tools to guide the operators when performing such tasks.

Our research work has studied the combination of two main strategies in order to manage this problem. The first strategy is the use of an audit mechanism that analyzes already deployed configurations, signals inconsistencies, and yields consistent

J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia, S. Preda
IT TELECOM Bretagne CS 17607, 35576 Cesson-Sévigné, France

configurations. Through this mechanism, moreover, we can fold existing policies and create a consistent and global set of rules — easy to maintain and manage by using a single syntax [13]. The second strategy is the use of a refinement mechanism that guarantees the proper deployment of such rules into new systems, yet free of inconsistencies. These two research strategies have been implemented into a software prototype called MIRAGE (which stands for MIsconfiguRAtion manaGEr).

Developed as a web service, MIRAGE can autonomously be executed under the control of several operators in order to offer the system with the following functions: (1) an intra-component analysis which detects inconsistencies between rules within single security component policies [11]; (2) an inter-component analysis of rules to detect inconsistencies between configurations of different devices [14]; (3) an aggregation mechanism to fold all the existing policies into a single, and consistent, global set of rules [13]; and (4) a refinement process to properly deploy the global set of rules over new security components [17, 18]. In all four cases, MIRAGE utilizes a description of the topology of the whole security architecture.

The use of MIRAGE can highly benefit the maintenance of multihomed autonomous systems. It might remain connected to the network and provide assistance to the complete set of component in the event of configuration maintenance or redeployment of configuration to face events such as detection of malicious activity or failures. In this sense, the refinement mechanism offered by MIRAGE guarantees that the set of rules deployed over the different components of a system is always consistent, not redundant, and optimal [19].

Paper Organization — Section 2 reviews recent results implemented in the MIRAGE prototype for guaranteeing correctness and consistency on single and distributed network security policies. Section 3 compares these approaches implemented in MIRAGE with other solutions proposed in both the science and the industry community. Section 4 closes the paper.

2 MIRAGE Prototype

MIRAGE is a management tool for guaranteeing the correctness and the consistency of configuration rules on single and distributed network security policies. It implements an analysis of components' configurations (i.e., configurations of firewalls, NIDSs, and VPN routers) to detect anomalies on their deployment. To do so, MIRAGE implements four main functions: intra-component analysis for the detection of inconsistencies between configuration rules within single security component policies; inter-component analysis of rules to detect inconsistencies between configurations of different devices; aggregation of policies for the creation of a consistent and global set of rules; and refinement mechanism for the deployment of global policies over the different components of new systems. We address in the sequel the key aspects of some of these functionalities implemented in the current version of our prototype.

2.1 Bottom-Up Analysis of Network Configurations

We assume here that a security policy has been empirically deployed into the network based on security administrator expertise and flair. It is then advisable to analyze the security rules deployed to detect and correct policy inconsistencies. These inconsistencies are often the origin of security holes exploited by dishonest parties. MIRAGE addresses this process and provides a discovery of inconsistencies and redundancies from component configurations. This process is presented based on two different schemes: (1) single- and (2) multi-component analysis.

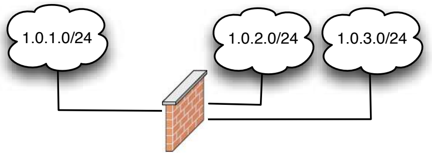
Single-component Analysis

MIRAGE provides a deterministic process to detect inconsistencies in the configuration of security components. It considers that these devices are configured in a standalone manner, using a set of configuration rules (e.g., filtering rules in the case of a firewall; and alerting rules in the case of a NIDS). A general configuration rule is defined as follows:

$$R_i : \{condition_i\} \rightarrow decision_i \qquad (1)$$

Regarding the previous expression, i is the relative position of a rule in the set, $\{condition_i\}$ is a conjunctive set of condition attributes such that $\{condition_i\}$ equals $A_1 \wedge A_2 \wedge ... \wedge A_p$ – being p the number of attributes of the given rule – and *decision* is a boolean value in $\{true, false\}$. For example, the decision of a filtering rule is positive ($true$) when it applies to a specific value related to *deny* the traffic it matches; and negative ($false$) when it points to *accept* the traffic it matches. Similarly, the decision field of an alerting rule is positive ($true$) when it applies to a specific value related to *alert* about the traffic it matches; and negative ($false$) when it applies to a specific value related to *ignore* the traffic it matches. Based on the sample scenario depicted by Figure 1, and its associated set of rules, we define the following set of anomalies detected by the intra-component audit process.

- *Intra-component Shadowing* — A configuration rule R_i is shadowed in a set of configuration rules R when such a rule never applies because all the packets that



R_1: {tcp,1.0.1.[10,20]:any, 1.0.1.[50,60]:any} → false
R_2: {tcp,1.0.2.[10,255]:any, 1.0.3.[0,255]:any} → false
R_3: {tcp,1.0.2.[1,30]:any, 1.0.1.[20,45]:any} → true
R_4: {tcp,1.0.2.[20,60]:any, 1.0.1.[25,35]:any} → false
R_5: {tcp,1.0.2.[30,70]:any, 1.0.1.[20,45]:any} → false
R_6: {tcp,1.0.2.[15,45]:any, 1.0.1.[25,30]:any} → true

Fig. 1 Single filtering policy scenario.

R_i may match, are previously matched by another rule, or combination of rules, with higher priority. E.g., rule R_6 is shadowed by the overlapping of rules $R_3 \cup R_5$.

- *Intra-component Redundancy* — A configuration rule R_i is redundant in a set of configuration rules R when the following conditions hold: (1) R_i is not shadowed by any other rule or set of rules; (2) when removing R_i from R, the security policy does not change. E.g., rule R_4 is redundant to $R_3 \cup R_5$.
- *Intra-component Irrelevance* — A configuration rule R_i is irrelevant when (1) source and destination addresses are within the same zone or (2) the component does not appear in the minimal route that connects the source zone (i.e., the rule matches traffic that never reaches the component). E.g., rule R_1 is irrelevant since both source and destination are in network $1.0.1.0/24$. Rule R_2 is also irrelevant since the filtering device is not part of the minimal route between networks $1.0.2.0/24$ and $1.0.3.0/24$.

The reader can find in [11, 14] the algorithms that enable MIRAGE the detection of the inconsistencies presented in this section, as well as correctness and computational complexity of the algorithms. Although we show that the theoretical complexity of the algorithms is very high, we show with a series of experimentations (cf. [14], Section 6) that we are always very far from the worst case. Indeed, only few attributes, such as source and destination addresses, may significantly overlap and exercice a bad influence on the algorithms complexity. Other attributes, such as the protocol or the port numbers, are generally equal or completely different when combining configuration rules. Moreover, when anomalies are discovered, some rules are removed – which significantly reduces the algorithms complexity.

Multi-component analysis

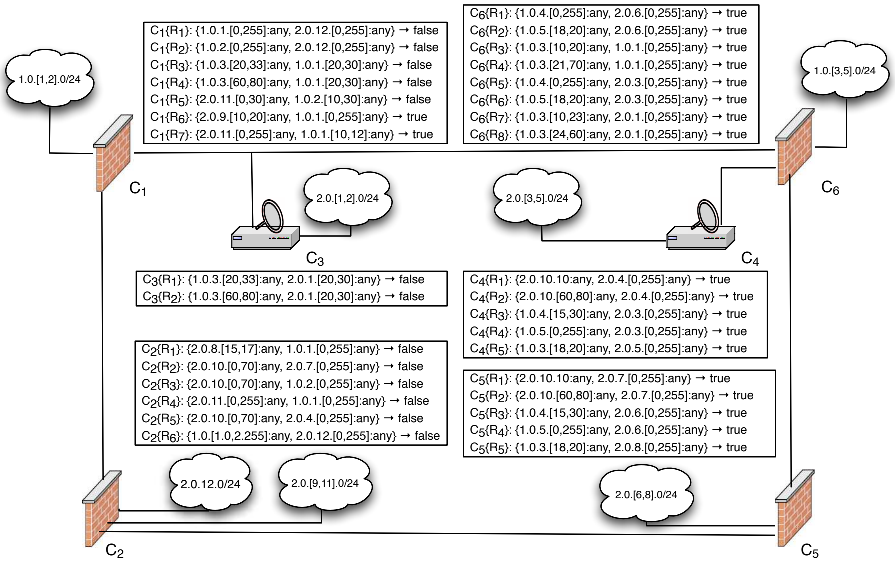
MIRAGE provides a second audit process to analyze multi-component setups (e.g., distributed architectures with firewalls and NIDSs in charge of multiple network security policies). In this sense, it can assume, for instance, that the role for detecting and preventing network attacks is assigned to several components. It will, then, look for inconsistencies hidden in their configurations. The detection process is based on the similarity between the parameters of configuration rules such as filtering an alerting rules. It checks, indeed, if there are errors in the configurations by comparing the policy deployment over each component that matches the same traffic. Based on the sample scenario depicted by Figure 2, we show in the sequel an example of the kind of inconsistencies detected by the inter-component audit process of MIRAGE.

- *Inter-component Shadowing* — A shadowing anomaly occurs between two components when the following conditions hold: (1) The component that is located closest to the origin of the traffic is a filtering device (e.g., a firewall); (2) The component where the anomaly is detected does not block or report (completely or partially) traffic that is blocked (explicitly, by means of positive rules; or implicitly, by means of its default policy), by the first component in the path (closest to the source). The following table shows some examples.

Rules	**Anomaly**
$C_6\{R_7\} \cup C_6\{R_8\}$ shadows $C_3\{R_1\}$	*full shadowing*
$C_6\{R_8\}$ partially shadows $C_3\{R_2\}$	*explicit partial shadowing*
Close policy of C_2 shadows $C_1\{R_5\}$	*implicit full shadowing*

- *Inter-component Redundancy* — A redundancy anomaly occurs between two components when the following conditions hold: (1) The component that is located closest to the origin of the traffic is a filtering device (e.g., a firewall); (2) The component where the anomaly is detected, blocks or reports (completely or partially) traffic that is already blocked by the first component. This kind of redundancy is often introduced by network officers expressly. It is important, however to alert about it, to warn the administrator that the rule has a special meaning (e.g., a message warning that if the rule applies, the upstream filtering devices are not working properly). The following table shows some examples.

Rules	**Anomaly**
$C_6\{R_1\}$ is redundant to $C_5\{R_3\}$	*full redundancy*
$C_6\{R_5\}$ is redundant to $C_4\{R_3\}$	*full redundancy*
$C_6\{R_2\}$ is redundant to $C_5\{R_4\}$	*partial redundancy*
$C_6\{R_6\}$ is redundant to $C_4\{R_4\}$	*partial redundancy*



```
C1{R1}: {1.0.1.[0,255]:any, 2.0.12.[0,255]:any} → false
C1{R2}: {1.0.2.[0,255]:any, 2.0.12.[0,255]:any} → false
C1{R3}: {1.0.3.[20,33]:any, 1.0.1.[20,30]:any} → false
C1{R4}: {1.0.3.[60,80]:any, 1.0.1.[20,30]:any} → false
C1{R5}: {2.0.11.[0,30]:any, 1.0.2.[10,30]:any} → false
C1{R6}: {2.0.9.[10,20]:any, 1.0.1.[0,255]:any} → true
C1{R7}: {2.0.11.[0,255]:any, 1.0.1.[10,12]:any} → true
```

```
C6{R1}: {1.0.4.[0,255]:any, 2.0.6.[0,255]:any} → true
C6{R2}: {1.0.5.[18,20]:any, 2.0.6.[0,255]:any} → true
C6{R3}: {1.0.3.[10,20]:any, 1.0.1.[0,255]:any} → true
C6{R4}: {1.0.3.[21,70]:any, 1.0.1.[0,255]:any} → true
C6{R5}: {1.0.4.[0,255]:any, 2.0.3.[0,255]:any} → true
C6{R6}: {1.0.5.[18,20]:any, 2.0.3.[0,255]:any} → true
C6{R7}: {1.0.3.[10,23]:any, 2.0.1.[0,255]:any} → true
C6{R8}: {1.0.3.[24,60]:any, 2.0.1.[0,255]:any} → true
```

```
C3{R1}: {1.0.3.[20,33]:any, 2.0.1.[20,30]:any} → false
C3{R2}: {1.0.3.[60,80]:any, 2.0.1.[20,30]:any} → false
```

```
C4{R1}: {2.0.10.10:any, 2.0.4.[0,255]:any} → true
C4{R2}: {2.0.10.[60,80]:any, 2.0.4.[0,255]:any} → true
C4{R3}: {1.0.4.[15,30]:any, 2.0.3.[0,255]:any} → true
C4{R4}: {1.0.5.[0,255]:any, 2.0.3.[0,255]:any} → true
C4{R5}: {1.0.3.[18,20]:any, 2.0.5.[0,255]:any} → true
```

```
C2{R1}: {2.0.8.[15,17]:any, 1.0.1.[0,255]:any} → false
C2{R2}: {2.0.10.[0,70]:any, 2.0.7.[0,255]:any} → false
C2{R3}: {2.0.10.[0,70]:any, 1.0.2.[0,255]:any} → false
C2{R4}: {2.0.11.[0,255]:any, 1.0.1.[0,255]:any} → false
C2{R5}: {2.0.10.[0,70]:any, 2.0.4.[0,255]:any} → false
C2{R6}: {1.0.[1.0,2.255]:any, 2.0.12.[0,255]:any} → false
```

```
C5{R1}: {2.0.10.10:any, 2.0.7.[0,255]:any} → true
C5{R2}: {2.0.10.[60,80]:any, 2.0.7.[0,255]:any} → true
C5{R3}: {1.0.4.[15,30]:any, 2.0.6.[0,255]:any} → true
C5{R4}: {1.0.5.[0,255]:any, 2.0.6.[0,255]:any} → true
C5{R5}: {1.0.3.[18,20]:any, 2.0.8.[0,255]:any} → true
```

Fig. 2 Example of an inter-component setup.

- *Inter-component Misconnection* — A misconnection anomaly occurs between two components when the first one, located closest to the source, is a firewall that permits (explicitly, by means of negative rules; or implicitly, through its default policy) all the traffic, or just a part of it, that is then denied by the component where the anomaly is detected. The following table shows some examples.

Rules	**Anomaly**
$C_5\{R_1\}$ and $C_2\{R_2\}$ are misconnected	*full explicit misconnection*
$C_5\{R_2\}$ and $C_2\{R_2\}$ are misconnected	*partial explicit misconnection*
$C_1\{R_5\}$ and policy of C_2 are misconnected	*full implicit misconnection*
$C_1\{R_6\}$, $C_2\{R_1\}$, and policy of C_2	*partial implicit misconnection*

The reader can find in [14] the algorithms that enable MIRAGE the detection of the inconsistencies presented in this section, as well as correctness profs, computational complexity, and experimental results. The complete set of analyzed configuration can be aggregated into a single, and consistent, global set of rules by using the aggregation mechanism presented in [13]. This global policy is, in fact, the main source of information used by the refinement mechanism presented in the sequel.

2.2 Top-Down Refinement of Global Policies

A second approach to address the management of consistency and correctness of network policies is the use of refinement mechanisms. In this way, we can perform a downward deployment of rules by unfolding a global set of security policies into the configurations of several components and guaranteeing that the deployed configurations are free of anomalies. In [10], for example, we presented a refinement mechanism that uses a formal model for the generation of filtering rules by transforming general rules into specific configuration rules. We address in this section some functionalities addressed by MIRAGE in this sense.

2.2.1 Model-driven Policy Deployment

If manually carried out, the process of deploying network security policies is often errorprone. In fact, without the right structural knowledge of the policy, the deployment of conflicting security requirements becomes very likely. This highlights the necessity of a more structured policy expression, i.e., only a formalized expression of the security policy may guarantee an error-free security policy to be deployed, with no ambiguities, no inconsistencies, no redundancies and no unnecessary details. Thus MIRAGE considers that an access control model and a formalized security policy is the first step toward enforcing by refinement the security of the system.

MIRAGE takes full advantage of the OrBAC model (Organization Based Access Control) [1] which is an extension of RBAC [20]. OrBAC presents a high abstraction level and covers a large panel of security policies since it natively provides means to express both static requirements (i.e., they are enforced once and for all) and *contextual* requirements (i.e., dynamic requirements). The OrBAC notions of *role*, *activity*, and *view* and also *context* prove very useful: the complexity of both the system (tens of firewalls, NIDSs, and VPN routers) and the policy (static and dynamic security requirements) is no longer an issue. The *role* regroups *subjects* (concrete network entities), the *activity* – *actions* (network services), the *view* – *objects* (e.g., IP packets, network entities) on which the same rules apply respectively. The notion of *context* confers the possibility to address a larger variety and also a finer granularity of security requirements, it captures the *conditions* (e.g., environmental factors) in which the security requirements are enforced and met. The OrBAC *context* allows the specification of these conditions directly at an abstract policy level.

The policy refinement mechanism of MIRAGE is in fact a set of deployment algorithms which constitutes the *downward* process: an OrBAC error-free security policy is refined into packages of rules for each security device in the given network. The aim is the *correct* deployment of a security policy, and this is achievable if some specific *security properties* are verified at the end. Such properties guarantee that no intra- nor inter- component anomalies are introduced (cf. Section 2.1). The formal frame to design the refinement mechanism of MIRAGE is presented in [18]. The policy deployment algorithms are developed using the B Method [2], a theorem proving method. The B Method offers the means to cope with the issue of stating the *interesting* security properties: besides an appropriate modeling language for both the OrBAC policy and the system/network specifications, it allows a formal expression of the properties related to the management of the intra- and inter- configuration anomalies during the *downward* process. This is ensured by some B *invariants*. Thus, from the early stage of their B development (i.e., abstract B specification), the policy deployment algorithms of MIRAGE target the interesting security properties. Examples of security properties we took into account, and expressed as B invariants, in [18] are:

- *Completeness* — This property states that if the network path from a *subject* to an *object* is correctly computed (i.e., it exists and the security components belonging to this path have the right functionalities with respect to the current $context_i$) the security OrBAC rule *Is_permitted*(*subject*, *action*, *object*, $context_i$) may and will be deployed. Clearly, this property is closely related to the network's architecture and the assumption of connectedness in the network architecture is required.
- *All traffic are regulated by filtering components* — This property is verified if there is, at least, exactly one firewall or one IPS on the path between the current subject and object.
- *Integrity and confidentiality* — These two properties are related to the establishment of VPN tunnels. The verification starts at higher levels: the current OrBAC security rule should be defined with a *protected* context — meaning that the trafic filtered by the associated rules must be protected by the VPN tunnels. Then, if a path is computed between the subject and the object and a VPN tunnel can be established on this path, the integrity and confidentiality properties are verified.

The work in [18] presents a complete analysis of security properties. Some may be specified at higher levels [9] and some may be identified from specific security requirements. The MIRAGE deployment algorithms were formally proved with the assumption of a conflict-free OrBAC policy and of a correct system architecture, i.e., no lack of security functionalities in the security components placed on the shortest-paths. Hence, as long as the system embeds all necessary security functionalities, there are no concerns in deploying the policies. The refinement provided by MIRAGE is a certified algorithm for a reliable and automatic security policy deployment. It is, however, realistic to consider that sometimes the system lacks some necessary security functionalities. Our proposed solutions to this problem are presented in the sequel.

2.2.2 Context Aware Policy Deployment

The security policies become more and more contextual. (Re)deploying a contextual security policy depends on the security device functionalities: either (1) the devices include all functionalities necessary to handle a context and the policy is consequently deployed to ensure its automatic changes or (2) the devices do not have the right functionalities to interpret a contextual requirement in its entirety. MIRAGE proposes two solutions to cope with the issue of the (re)deployment of access control policies in a system that lacks the necessary functionalities to deal with contexts:

1. *Dynamic deployment*: MIRAGE considers a central entity (hereafter called *PDP* – Policy Decision Point) which (partially) manages some contexts.
2. *Optimization deployment*: if the previous solution does not stand.

Obviously, the OrBAC formalism is maintained. These two solutions presented in [17] and [19] respectively can then be jointly used whenever the security devices (hereafter called *PEPs* – Policy Enforcement Points) are not rich enough in functionalities so as to manage all contexts by themselves. In this way, a complete deployment of the (contextual) policy may be achieved.

The Methodology

Let SR = (Decision, Role, Activity, View, Ctx) be a security rule of the OrBAC policy P (SR $\in P$) and SR$'$ = (Decision, Role, Activity, View, Ctx$'$) with Decision $\in$ {Permission, Prohibition}. We call SR$'$ the SR *contextual version* over the context Ctx$'$. Let PEP$_i$ be an enforcement point able to manage only the context Ctx$'$ and SR be the security rule PEP$_i$ must enforce. We investigate how SR$'$ can be deployed and thus enforced by PEP$_i$ even if Ctx$'$ is not equal to the context Ctx of the initial rule SR to be deployed. The final aim is to deploy the SR rule and one of the following situations appears:

- *Case 1* — The PEP$_i$ manages the entire Ctx context. The rule SR is directly deployed over PEP$_i$ and the PDP does not manage SR anymore. Otherwise, the PDP has to manage a part of the Ctx context.

- *Case 2* — The PEP_i manages only Ctx_2, a part of the Ctx context. We note this case as Ctx_1 = Ctx - Ctx_2. The deployment is *dynamic* and the PDP manages Ctx_1: the PDP must deploy SR′, the SR contextual version over Ctx_2 on the PEP_i when Ctx_1 becomes active. Once Ctx_1 is deactivated, the PDP must be able to retrieve the deployed SR′ from PEP_i. For example and as suggested in [17], Ctx_1 may represent a threat context activated by the detection of some intrusion. Thus, *Case 2* provides means to dynamically redeploy the policy to face this intrusion. This represents the *dynamic deployment* solution.
- *Case 3* — Neither the PEP_i, nor the PEP_i and the PDP working together manage the Ctx context. Then, two solutions are possible: (I) there may be a context closely related with Ctx which is still managed by the system as described in *Case 2* (in fact, we refer to the OrBAC context *hierarchies*); or (II) if the system does not provide *hierarchies*, the last option is to find a security functionality closely-equivalent to the one necessary to handle the Ctx context. Both (I) and (II) represent the *best deployed policy* solution.

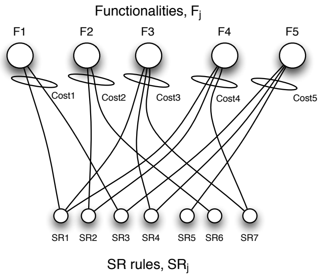
Dynamic Deployment

The formalization of this solution is based on the use of *ECA* (*Event Condition Action*) [6]. The algorithms running at the PDP level deploy (or retrieve) security rules over the PEPs when the contexts are activated (or deactivated). To be effective, this solution requires a specific communication protocol between the PDP and the PEPs. Actually our method uses the Netconf (cf. http://www.ops.ietf.org/netconf/) protocol with the Yencap (cf. http://ensuite.sourceforge.net/) open-source implementation which we adapted accordingly. Several performance tests were realized and presented in [17]. The results proved to be satisfactory.

Best Deployed Policy

There are scenarios in which the PDP and/or PEPs cannot entirely handle the Ctx context related to an SR rule. Instead of skipping such rules (with the result of, for example, a too restrictive deployed policy at the end), MIRAGE proposes the following solutions of: (1) finding a closely related context to the unmanaged one and which may be managed by the PDP and/or PEP and/or (2) finding a close enough functionality to deal with the unmanaged context if solution (1) does not apply. OrBAC proves very effective for the first solution since OrBAC provides *context hierarchies*. Thus, it is enough to find either the *more specialized* context than Ctx (to deploy permissions) or less specialized ones (to deploy prohibitions).

The second solution is solved with an optimization approach. The system presents no optimal functionality to manage the Ctx context but only *closely-equivalent* ones. We declare these functionalities with the *Close_Fs*() predicate. A notion of *cost* of using a given functionality to deploy certain rules in the Ctx context is introduced and the deployment problem is transformed into an optimization one. The result is a bipartite graph where the optimization solution is obtained with linear programming. Figure 3 depicts a proper example. Notice that, globally, the cost of deploying the SR rules over the Ctx contexts is minimized. The optimal functionalities necessary to handle the Ctx context will be substituted by closely-equivalent ones.



(a) The LP Problem.

Case A

minimize $\sum_{j=1}^{n} \mathrm{Cost}_j x_j$,

subject to $\sum_{j=1}^{n} d_{ij} x_j \geq 1,\ i \in (1 \ldots m)$

$x_j = 0 \text{ or } 1,\ j \in (1 \ldots n)$

where $x_j = 1$ if j is in FS, $x_j = 0$ otherwise; and $d_{ij} = 1$ if SR_i could be deployed using the functionality F_j, $d_{ij} = 0$ otherwise.

Case B

minimize $\sum_{j=1}^{n} \mathrm{Cost}_j x_j$,

subject to $\sum_{j=1}^{n} d_{ij} x_j = 1,\ i \in (1 \ldots m)$

$x_j = 0 \text{ or } 1,\ j \in (1 \ldots n)$

where $x_j = 1$ if j is in FS, $x_j = 0$ otherwise; and $d_{ij} = 1$ if SR_i could be deployed using the functionality F_j, $d_{ij} = 0$ otherwise.

(b) The LP Solutions.

Fig. 3 LP Problem and Solutions.

3 Related Works

A significant amount of work has been reported in the area of security management at network level in order to analyze and fix existing configurations. The most significant approach to firewall policy analysis is the one by Al-Shaer et al. (e.g., approaches presented in [4, 5]) which provides efficient solutions to detect policy anomalies in both single- and multi-firewall configuration setups. Their detection algorithms are based on the analysis of relationships between rules two by two. Therefore, errors due to the union of rules are not explicitly considered (as our approach does). Some workarounds can be provided to solve this situation. For instance, it is possible to break down the initial set of rules into an equivalent set of rules free of overlaps between rules. However, no specific algorithms were provided in [4, 5] to manage this solution. Another related work is the proposal presented in [21], which uses a model checking formalism for detecting inconsistencies and redundancies in single firewall configurations. This proposal handles the limitation pointed out in the works by Al-Shaer et al., by addressing directly the way how traffic is handled by the components. The complete set is divided in three main sets: traffic that is permitted, traffic that is prohibited, and traffic for which no rules apply. The proposal in [21], as well as other similar approaches, such as [16], only address intra-component analysis. Moreover, none of them have presented specific mechanisms for verifying policies other than filtering ones.

Regarding the analysis of VPN routers' configurations, the most significant approach compared to ours is proposed in [12]. The authors propose a technique that simulates VPN tunneling processing and reports any violation of the security policy requirements. In their approach, if an access rule concerning a protected traffic between two points is implemented by configuring more than one VPN overlapping tunnel, the risk is that in some network zones the IP packets circulate without any protection. The authors present a discovery process to detect such situations and propose a high-level language to deal with VPN policies. Although this approach can discover some violations in a certain simulation scenario, there is no guarantee that it discovers every possible violation that may exist. In addition, the proposed technique only discovers VPN conflicts resulting from incorrect tunnel overlap, but does not address the other types of conflicts. Another attempt to deploy VPN configurations free of anomalies is [7] where the authors propose a central-entity approach with high level language conflict resolution techniques also - the algorithms remained however unevaluated. However, a significant aspect is ignored in both previous approaches: the security policy cannot be seen as two independent sets of requirements (i.e., VPN tunnels and firewalls modeled separately). The use of a single access control model in our approach solves this limitation and allows us to deal with a global set of security requirements and to address inter-mechanisms anomalies (e.g., firewall vs. VPN conflicts) at the same time.

Another significant approach compared to ours is the RBAC-based proposal presented in [8], called Firmato. This new solution aims at configuring filtering devices following an approach of separation between the security policy model and the technology specifications. It, therefore, ensures policy deployment independently of the network topology. The tool is based on an *Entity — Association* model (abstract level) which takes into account the network topology as a role. The instantiation of the model is based on a specific language that allows a downward transformation of the global policy into a set of firewall configurations. However, the use of the role concept used in Firmato, which defines the network capabilities, becomes ambiguous in its semantics. The authors use the notion of *group* to handle this situation. A group can identify, in fact, a set of hosts but also a role or a set of roles. Its use does not ensure, indeed, a clear separation between the network level and the security policy, making difficult the use of this tool to model complex networks. The authors use, moreover, privilege inheritance through group hierarchies in order to derive permissions. If permission inheritance is related to the so-called *open group*, prohibitions are inherited through a *close group*. The notion of group clearly introduces ambiguities and seems to be useless at this abstraction level.

Support tools can also be used to assist administrators in their task of configuring security devices. Proper examples are the LogLogic Security Change Manager (cf. http://loglogic.com/products/), formerly known as Solsoft Policy Server and Network Security Policy Server, the Firewall Builder (cf. http://fwbuilder.org/), CheckPoint SmartCenter (cf. http://checkpoint.com/products/smartcenter/), Juniper Network and Security Manager (cf. http://www.netutils.com/), and the Cisco Security Manager (cf. http://cisco.com/go/csmanager). In a relatively high level language, these support tools allow the configuration of different vendors' devices and support the security administrators in the deployment of large configurations on heterogeneous networks. We observe the following problems when using such tools. First, they do not offer a semantic model rich enough to express a global security policy. Although it is possible to define variables, and thus to define access rules involving such variables, the administration tasks are not much simplified. The security officer always needs a global view of the topology in order to correctly assign each rule to network devices; then, there is no automatic discovery of security devices that optimally implement an access rule involving an IP source and a destination. Furthermore, the lack of a real downward approach like ours is partially replaced by other tools (e.g., Cisco conflict discovery tools) that need the security officer's assistance and that unfortunately only guarantee conflict resolution for local configurations.

4 Conclusions

We addressed the managing of network security policies free of anomalies or inconsistencies. Two main approaches were presented: (1) the use of bottom-up process to detect and fix configuration errors over components already deployed; and (2) the use of a top-down process to perform an automatic deployment of component configurations free of inconsistencies. The implementation of these two approaches in a software prototype demonstrates the practicability of our work. We finally compared the functionality of MIRAGE with some other solutions proposed in both the science and the industry community, and showed some advantages of our approaches. As future work, it is expected to add new a feature in MIRAGE to manage the update of components' configurations. This new feature will guide the operators to determine the impact that the removal or the addition of new configuration rules in the system might suppose. It is also expected to give support to determine dynamic tuning of configurations. In this case, the new feature is expected to compare and test the equivalence between different configurations. For example, the security operator can verify whether the new settings of a new configuration setup will perform well enough, and in compliance with the global security policy. Finally, it is also intended to complement the upward and the downward approaches offered by MIRAGE with an automatic discovery of roles associated with different security components already deployed in the system. It is planned the use of role mining techniques, for example, to analyze existing access control roles associated to the components (to derive, after the analysis, the appropriate rules of the global configuration).

Acknowledgments – This work has been supported by a grant from the Brittany region of France and by the following projects: POLUX ANR-06-SETIN-012, SEC6 Project, TSI2007-65406-C03-03 E-AEGIS, and CONSOLIDER CSD2007-00004 "ARES".

References

1. A. Abou el Kalam, R. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel and G. Trouessin. Organization Based Access Control. IEEE 4th Intl. Workshop on Policies for Distributed Systems and Networks, pp. 120–131, Lake Come, Italy, 2003.
2. J. R. Abrial. The B-Book — Assigning Programs to Meanings. Cambridge University Press, ISBN 052149619-5, 1996.
3. H. Adiseshu, S. Suri, and G. Parulkar. Detecting and resolving packet filter conflicts. *Joint Conference of the IEEE Computer and Communications Societies*, pp. 1203–1212, 2000.

4. E. S. Al-Shaer and H. H. Hamed. Discovery of Policy Anomalies in Distributed Firewalls. In *IEEE INFOCOM'04*, March, 2004.
5. E. S. Al-Shaer and H. H. Hamed. Taxonomy of Conflicts in Network Security Policies. In *IEEE Communications Magazine*, 44(3), March, 2006.
6. C. Baral, J. Lobo, and G. Trajcevski. Formal Characterization of Active Databases. *5th International Conf. on Deductive and Object-Oriented Databases*, 1997.
7. S. Baek, M. Jeong, J. Park, T. Chung. Policy based Hybrid Management Architecture for IP-based VPN. *In Network Operations and Management Symposium*, NOMS 2000.
8. Y. Bartal, A. Mayer, K. Nissim, A. Wool. Firmato: A Novel Firewall Management Toolkit. In *IEEE Symposium on Security and Privacy*, pp. 17–31, Oakland, California, May, 1999.
9. N. Benaissa, D. Cansell, and D. Méry. Integration of Security Policy into System Modeling. *7th International B Conference*, LNCS, vol. 4355, pp. 232–247, France, 2007.
10. F. Cuppens, N. Cuppens, T. Sans, and A. Miège. A formal approach to specify and deploy a network security policy *Second Workshop on Formal Aspects in Security and Trust*, Toulouse, France, August 2004, pp. 203–218.
11. F. Cuppens, N. Cuppens, and J. Garcia-Alfaro. Misconfiguration management of network security components. *7th International Symposium on System and Information Security (SSI 2005)*, Sao Paulo, Brazil, November 2005, pp. 1–10.
12. Z. Fu, et al. IPSec/VPN Security Policy: Correctness, Conflict Detection and Resolution. *International Policy Workshop*. January 2001.
13. J. Garcia-Alfaro, F. Cuppens, and N. Cuppens. Aggregating and Deploying Network Access Control Policies. *2nd International Conference on Availability, Reliability and Security (ARES 2007)*, IEEE Computer Society, 532-539, Vienna, Austria, April 2007.
14. J. Garcia-Alfaro, F. Cuppens, and N. Cuppens. Complete Analysis of Configuration Rules to Guarantee Reliable Network Security Policies. *International Journal of Information Security*, Springer, 7(2):103–122, April 2008.
15. P. Gupta. Algorithms for routing lookups and packet classification. Ph.D. Dissertation, Stanford University, Department of Computer Science, 2000.
16. A. X. Liu and M. G. Gouda. Complete Redundancy Detection in Firewalls. In *19th Annual IFIP Conference on Data and Applications Security (DBSec-05)*, pp. 196–209, Storrs, Connecticut, August, 2005.
17. S. Preda, F. Cuppens, N. Cuppens-Boulahia, J. Garcia-Alfaro, L. Toutain, and Y. Elrakaiby. A Semantic Context Aware Security Policy Deployment. *ACM Symposium on Information, Computer and Communications Security*, pp. 251–261, Sydney, Australia, March 2009.
18. S. Preda, N. Cuppens-Boulahia, F. Cuppens, J. Garcia-Alfaro, and L. Toutain. Model-driven Security Policy Deployment: Property Oriented Approach. *International Symposium on Engineering Secure Software and Systems (ESSoS10)*, LNCS, Springer, February 2010.
19. S. Preda, N. Cuppens-Boulahia, F. Cuppens, and L. Toutain. Architecture-Aware Adaptive Deployment of Contextual Security Policies. *Fifth International Conference on Availability, Reliability and Security (ARES 2010)*, IEEE Computer Society, February 2010.
20. R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, 1996.
21. L. Yuan, J. Mai, S. Su, H. Chen, C. Chuah, and P. Mohapatra. FIREMAN: a toolkit for FIREwall Modeling and ANalysis. In *IEEE Symposium on Security and Privacy*, pp. 199–213, Oakland, California, 2006.