An Approach to Information Security Policy Modeling for Enterprise Networks

Dmitry Chernyavskiy, Natalia Miloslavskaya

An Approach to Information Security Policy Modeling for Enterprise Networks

Dmitry Chernyavskiy and Natalia Miloslavskaya

Information Security of Banking Systems Department
National Research Nuclear University MEPhI (Moscow Engineering Physics Institute)
31 Kashirskoe shosse, 115409, Moscow, Russian Federation
D.S.CH@mail.ru, NGMiloslavskaya@mephi.ru

Abstract. Network security management is one of the most topical concerns of information security (IS) in modern enterprises. Due to great variety and increasing complexity of network security systems (NSSs) there is a challenge to manage them in accordance with IS policies. Incorrect configurations of NSSs lead to outages and appearance of vulnerabilities in networks. Moreover, policy management is a time and resource consuming process, which takes significant amount of manual work. The paper discusses issues of policy management process in its application for NSSs and describes a policy model aimed to facilitate the process by means of specification of IS policies independently on platforms of NSSs, selection of the most effective NSSs aligned with the policies, and implementation of the policies in configurations of the NSSs.

Keywords: Information Security Policy, Policy Management Process, Network Security System, Finite Automaton, Algebra.

1 Introduction

Network security in most enterprises relies on such network security systems (NSSs) as firewalls and intrusion detection/prevention systems (IDS/IPS) [1]. However, management of NSSs faces challenges tied with time-consuming manual processes, lack of visibility in information security (IS) policies and configuration errors, which lead to network outages and appearance of vulnerabilities [2]. For instance, a policy (hereafter "policy" means "IS policy") for Check Point or Cisco firewalls may consist of thousands of rules and such complexity of policies is the main cause of configuration errors [3,4].

Thus, on the one hand, increasing number of NSSs and their increasing functionality allow to counter more threats and reduce IS risks as a result. On the other hand, complexity of NSSs' management leads to new risks and time-consuming processes, which reduce overall efficiency of NSSs utilization. Therefore, policy management process for NSSs needs simplification in order to reduce probability of errors and efforts on time-consuming tasks.

A formal approach to policy modeling presented in the paper is aimed to facilitate the process by means of specification of policies independently on platforms of NSSs,

selection of the most effective NSSs, and translation of the specified policies into configurations of the NSSs. The contributions of the paper are (a) a policy model for NSSs based on a finite automaton representation of an NSS, (b) an approach to classification of NSSs and selection of the most effective NSS, and (c) a policy algebra based on the model.

The paper is organized as follows. Section 2 overviews policy management process for NSSs and discusses related work on policy modeling. Basic notions of the policy model for NSSs and the approach for selection of the most effective NSS are presented in Section 3. Section 4 introduces the policy algebra for NSSs. Finally, Section 5 concludes the paper.

2 Related Work

Models of policy management process are presented in [5,6,7,8]. All the models consider policy management as iterative process and include similar operations. The most detailed description of policy management process is presented in [5] and from the standpoint of NSSs management the following operations of the process are important. During Policy Assessment step a request for initial policy creation or update of the existing one is evaluated in order to identify policy conflicts and effects. The requested change should be made in the framework of existing IS maintenance system (ISMS includes IS management system and security tools and measures). Identification of IS threats for assets and a list of appropriate options of NSSs that counter the threats, payroll and non-payroll cost of the options as well as determination of options priority are included in Risk Assessment step. Creation of new policies or update of existing ones proceeds during Policy Development step. Requirements for ISMS are derived upon Requirements Definition step in order to assure that it is aligned with new policies. In the course of Controls Definition step the requirements to ISMS are transformed into a selection of the best options of NSSs and requirements to them. Upon Controls Implementation step the NSSs are installed and configured in accordance with the policies. Compliance and audit checks carried out during Monitor Operations step in order to ensure that ISMS functions in alignment with the policies. Review Trends and Manage Events step includes identification of events and trends (internal and external in relation to an enterprise) that may indicate a need to make changes in the policies. Further, during the step possible changes are evaluated against any appropriate criteria in order to make sure that the changes are essential and escalated to the beginning of the process [5]. In addition, if during Policy Assessment, Risk Assessment or Policy Development steps it is identified that some policies are not needed any more, then they must be retired [7]. Note that policy management process is iterative due to continuous changes in technologies, business environment and legal requirements [8].

Thus, in the scope of the process there are the following significant groups of tasks: (a) development and update of policies, (b) selection of the best options of NSSs that are in line with the policies, (c) translation of the policies into configurations of the selected NSSs, and (d) detection and resolution of conflicts in the policies. Formal policy models can be applied to the process in order to automate the tasks and as a result

reduce time-consuming efforts and probability of errors. Policy modeling approaches are extensively discussed in literature, some of them are presented in [9,10,11,12,13].

The modeling method [9] is based on four independent atomic functionalities (end-flow, channel, transform, and filter functionalities) and a formal language that allows specification and validation of policies for the functionalities. However, such functionalities as logging and alerting are not presented in the model. The model also lacks means for selection of the best options of functionalities.

An access-control language based on XML syntax and Organization-Based Access Control (OrBAC) model, which is an extension of Role-Based Access Control (RBAC), is given in [11] and intended to specify firewall policies. While OrBAC supports definition of obligation policies and contexts, the language, however, does not include these capabilities and deals only with authorization policies. Another limitation is consideration only of IP addresses, protocols and ports as filtering parameters in the policies, which does not allow specification of more sophisticated filtering policies that inspect other parameters of network packets.

Another method based on OrBAC is described in [11]. The method supports definition of different contexts and obligations within relevant contexts. However, when a policy cannot be implemented by the existing security architecture, the method does not provide means to select appropriate security functionalities for the policy.

An approach [12, 13] provides specification of OrBAC policies and contexts as well as derivation of configurations for security functionalities. It also allows selection of the best options of functionalities in terms of their cost. For this purpose the approach introduces a notion of "closely equivalent" functionalities but lacks any formal criterion to identify such functionalities.

A formal framework presented in [14] provides means for synthesizing secure network configurations. The framework utilizes a network topology, security requirements and business constraints such as usability and budget limits as inputs in order to derive a cost-effective placement of NSSs in the topology. The framework uses policy requirements only as inputs, but does not include capabilities for specification of policies and their transplantation into configurations of NSSs. Functionalities of NSSs within the framework are considered as "primitive isolation patterns", which can be composed into "composite isolation patterns"; however, a formal classification criterion of the functionalities is not presented.

Hence, the considered approaches either do not support specification of all types of policies for NSSs or do not provide means for selection of the best options of NSSs.

3 A Policy Model

A system is called an *NSS* if it is intended to directly or indirectly secure information transferable through an enterprise's network. Assume that function of an NSS is to form any output in accordance with a policy by means of processing of network traffic that comes to its input. In the general case an output of an NSS is network traffic or messages such as log entries and alerts that sent to other systems or IS administrator's console.

Let P be a set of all possible policies that NSSs can implement. Actually, P consists of sequences of symbols that form commands for different NSSs. If any NSS uses GUI instead of CLI, its policy can be expressed as a text string. For instance, the first rule of Check Point firewall policy shown on Fig. 1 can be written as *"N=1 Source=Any Destination=Web-Server Service=Any …"* or in any other way that reflects semantics of the rule. Also, the set P includes the empty sequence ε. Let T be a set of network traffic, where $t \in T$ is a network packet (i.e., a sequence of bits), or the empty sequence ε, which means absence of traffic. Consider an NSS as a finite automaton:

$$F = < T \times P, S, T \times M, \delta, f >,$$

where $T \times P$ is an input alphabet, S is a finite set of internal states of NSS, $T \times M$ is an output alphabet, $\delta: T \times P \times S \to S$ is a state-transformation function and $f: T \times P \times S \to T \times M$ is an output function, M is a set of output messages, which also includes ε (similar to the case of the set T). An NSS functions in discrete time τ and transforms an input traffic $t(\tau) \in T$ into an output traffic $t'(\tau) \in T$ in accordance with a policy $p(\tau) \in P$. An NSS changes its internal state $s(\tau) \in S$ into state $s(\tau + 1) \in S$ while it functions (Fig. 2). The set S can include such parameters as time, number and sequence of packets in a session and other parameters essential to model stateful analysis of a network traffic. Such NSS as a stateless packet filter can be considered as an NSS with one state, i.e., $|S| = 1$.

NO.	SOURCE	DESTINATION	VPN	SERVICE	ACTION	TRACK	INSTALL ON	TIME	COMMENT
1	* Any	Web-Server	Any Traffic	http	accept	None	Corporate-gw	* Any	Allow connections to corporate web server
2	Internal-net	SQL-Server	Any Traffic	MS-SQL	accept	None	Corporate-gw	* Any	Allow connections from internal network to corporate database server
3	Admin-subnet	Management	Any Traffic	CPMI	accept	None	Corporate-gw	* Any	Allow connections from administrator's network to Check Point management server
4	* Any	* Any	Any Traffic	* Any	drop	None	Corporate-gw	* Any	Drop all other traffic

Fig. 1. An example of Check Point firewall policy

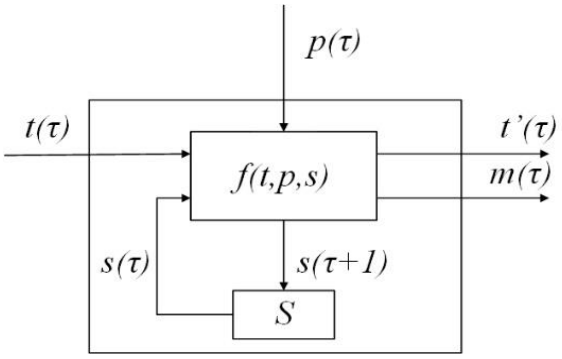

Fig. 2. A finite automaton model of an NSS

Assume that every policy $p \in P$ is represented as a triple of the following finite vectors:

- $\vec{x} = (x_1, x_2, \dots)$ is an input vector, describing an input traffic of an NSS, where $x_i \in X_i$, while X_i is a set of parameters of any homogenous nature (for instance, X_i can be a set of IP addresses, protocols, port numbers or any other attribute of a network traffic);
- $\vec{y} = (y_1, y_2, \dots)$ is an output vector, describing an output traffic and/or messages generated by an NSS, were $y_i \in Y_i$ while Y_i is a set of parameters of output network traffic of any homogenous nature (similar to the case of the input vector) or a set of parameters of the messages;
- $\vec{z} = (z_1, z_2, \dots)$ is a state vector, describing an internal state of an NSS. For instance, $z_i \in Z_i$ can be a system time of an NSS.

Any policy $p \in P$ is said to be *definite* if the triple of the policy consists of explicitly defined parameters (concrete protocols, IP addresses, ports, messages, etc.). Any policy that consists of parameters expressed in general view without concretization of elements of corresponding sets (IP address, protocol or port in general, etc.) it is called a *generalized* policy. Note that all policies from the set P are definite policies, since only definite policies are implementable by NSSs. By P_{gen} denote a set of generalized policies. For instance, vector $\vec{x} = (192.168.1.1, TCP, 80)$ belongs to a definite policy, while $\vec{x}_{gen} = (Source\ IP, Protocol, Port)$ is the vector of corresponding generalized policy, where Source IP is the set of source IP addresses, Protocol is the set of protocols, Port number is the set of ports. For any generalized policy $p_{gen} \in P_{gen}$ there is a corresponding subset P' of definite policies from the set P, this subset is denoted by $Def(p_{gen})$. Let P_{sub} be the set of subsets of P, then $Def(p_{gen})$ is the injective map $Def: P_{gen} \to P_{sub}$.

The input, output, and state vectors of any generalized policy p are denoted by $X(p)$, $Y(p)$, and $Z(p)$, respectively. Let $\leq$ be the partial order relation on P_{gen} such that $p \leq p'$ if and only if all parameters of the vectors of the generalized policy p are included in the respective vectors of the generalized policy p':

$$p \leq p' \leftrightarrow X(p) \subseteq X(p'), Y(p) \subseteq Y(p'), Z(p) \subseteq Z(p').$$

Let $\mathcal{F}$ be a set of all existing NSSs. The fact that an NSS $F \in \mathcal{F}$ is capable to implement a policy $p \in P$ or a subset of policies $P' \subseteq P$, i.e., p or P' is included to definitional domain of the state-transformation function δ and the output function f of F, is denoted by $F(p)$ and $F(P')$ in the case of a policy and a subset of policies respectively. Any $F \in \mathcal{F}$ is defined on the entire set T and in the general case on some subset $P' \subseteq P$, where P' can be expressed as the finite union:

$$P' = Def(p_{gen1}) \cup Def(p_{gen2}) \cup \dots \cup Def(p_{genk}).$$

An NSS $F(Def(p_{gen})) \in \mathcal{F}$ is said to be *simple* if there does not exist an NSS $F' \in \mathcal{F}$ that implements any policy with excluded parameters:

$$F(Def(p_{gen})) \in \mathcal{F}_s \leftrightarrow \nexists\, F'(Def(p'_{gen})) \in \mathcal{F}: p'_{gen} \leq p_{gen} \wedge p'_{gen} \neq p_{gen},$$

where $\mathcal{F}_s$ is the set of simple NSSs. In other words, if exclusion of any subset of parameters from a policy leads to meaningless policy that cannot be implemented by any NSS, then the NSS is simple. Let $P_s = \{p_{gen} \in P_{gen}: \exists F(Def(p_{gen})) \in \mathcal{F}_s\}$ be the set of generalized policies that corresponding definite policies can be implemented by simple NSSs. If an NSS is not simple, it is called *composite*. Simple NSSs can be combined parallel or sequentially with the purpose of constructing a composite NSS.

For example, when considering Check Point firewall policy (Fig. 1) and network address translation (NAT) policy (Fig. 3), the NSSs can be decomposed to simple NSSs that implement single rules. Fig. 4 shows an example of the composite NSS that implements mentioned firewall and NAT policies. Note that the latter rule of the firewall policy in Fig. 1 is not presented in Fig.4 because each simple NSS F_{fw} blocks by default any traffic that does not match its policy (i.e., it outputs the empty sequence ε). In contrast with firewall policies, NAT policy accepts any traffic by default, therefore an additional simple NSS F_{nat} with the default policy (which accepts all traffic that does not match the other NAT rules) is introduced into the model in Fig. 4. Note also that state blocks S (shown in Fig. 2) of each simple NSS are not depicted in Fig. 4 for the sake of compactness.

Two simple NSSs $F_1(Def(p_{gen1})), F_2(Def(p_{gen2})) \in \mathcal{F}_s$ are called *equivalent* if and only if they produce equal outputs for equal inputs while implementing policies:

$$F_1\left(Def\left(p_{gen1}\right)\right) \cong F_2\left(Def\left(p_{gen2}\right)\right) \leftrightarrow$$

$$\forall p_1 \in Def\left(p_{gen1}\right) \exists p_2 \in Def\left(p_{gen2}\right): f_1^*(t, p_1, s_1) = f_2^*(t, p_2, s_2)\ \forall t \in T^* \ \wedge$$

$$\forall p_2 \in Def\left(p_{gen2}\right) \exists p_1 \in Def\left(p_{gen1}\right): f_2^*(t, p_2, s_2) = f_1^*(t, p_1, s_1)\ \forall t \in T^*$$

where s_1 and s_2 are initial states of F_1 and F_2 respectively, T^* is a set of finite sequences of network packets, f_1^* and f_2^* are extensions of f_1 and f_2 to T^*.

By this equivalence relation the set of simple NSSs partitioned to equivalence classes. All simple NSSs inside any equivalence class produce equal outputs while implementing respective policies and processing network traffic. However, their policies from the syntax point of view can be different.

In order to demonstrate the equivalence of NSSs consider Check Point and Cisco firewalls along with the following policy: *"Hosts from the network 192.168.1.0/24 are allowed to establish connections to 80 TCP-port on server 10.1.1.10. Connection attempts must be logged".* Note that the policy consists of two parts: authorization (allows connections to the server) and obligation (requires logging of connection attempts). Each of two selected NSSs are able to implement the policy. It can be represented in Check Point as shown in Fig. 5. In order to implement the policy in Cisco it is necessary to add one rule to an access-list (for instance, access-list 101):

```
access-list 101 permit tcp 192.168.1.0 255.255.255.0 host
10.1.1.10 eq 80 log
```

NO.	ORIGINAL PACKET			TRANSLATED PACKET			INSTALL ON	COMMENT
	SOURCE	DESTINATION	SERVICE	SOURCE	DESTINATION	SERVICE		
1	Web-Server	* Any	* Any	Web-Server	= Original	= Original	Corporate-gw	Automatic rule (see the network object data).
2	* Any	Web-Server	* Any	= Original	Web-Server	= Original	Corporate-gw	Automatic rule (see the network object data).

Fig. 3. An example of Check Point NAT policy

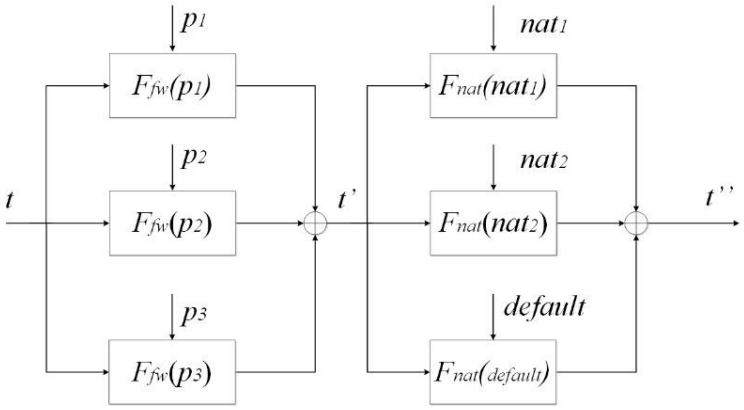

Fig. 4. A composition of simple NSSs

NO.	SOURCE	DESTINATION	VPN	SERVICE	ACTION	TRACK	INSTALL ON	TIME	COMMENT
1	Network-192.168.1.0_24	Server-10.1.1.10	Any Traffic	TCP http	accept	Log	Corporate-gw	* Any	

Fig. 5. Check Point firewall policy

The input vectors of generalized policies for Check Point and Cisco are equal: (Source Address, Destination Address, Protocol, Destination Port). Note that the field "service" in Check Point policy (Fig. 5) implies protocol and destination port. Output traffic is the same as an input and it is expressed with keywords *accept* and *permit* in the case of Check Point and Cisco respectively. In addition, a log message must be generated that is expressed with keyword *log* in both cases. Thus, the output vectors are equal for both NSSs: (Accept, Log). State vectors of Check Point and Cisco policies are (VPN, Time, Comment) and (Access-list Number) respectively. However, the parameters of Check Point's state vector are not initialized and consequently do not affect the policy. Access-list number of Cisco is not essential in this case either. Semantics of these two rules are equal and hence the NSSs are equivalent. Undoubtedly, Check Point and Cisco firewalls are not equivalent in general, but abstractions of them (i.e., simple NSSs) that implement the considered policies are equivalent.

The approach to classification of NSSs consists of the following high-level steps:

- decomposition of an NSS to simple NSSs;
- sorting out the triple of vectors of the generalized policy for each simple NSS;
- search for an equivalence class for each simple NSS:

— if there exists a class such that a simple NSS is equivalent to, then it must be added to the class;
— if there is no appropriate equivalence class for a simple NSS, then it forms its own class.

The classification facilitates evaluation of an NSS $F \in \mathcal{F}$ by means of assigning the following rating R_F:

$$R_F = \sum_i W_{F_i} I_{F_i}^F,$$

where $W_{F_i} > 0$ is a weight of a simple NSS F_i (for instance, the number of IS threats that the NSS counters), which can be calculated by an expert evaluation, and $I_{F_i}^F \in \{0,1\}$ is an indicator that shows whether an NSS F is included into equivalence class F_i.

The classification of NSSs built in terms of the described equivalence helps to find NSSs with required functions. For instance, if it is necessary to use functions F_a, F_b and F_c in some node of an enterprise's network in order to implement particular policy, then an NSS that is simultaneously included in equivalence classes F_a, F_b, and F_c can be selected for this purpose. If there are more than one NSS with required functions then the most effective solution among them is the one that has maximum rating-cost ratio:

$$\frac{R_F}{C_F} \to max,$$

where C_F is sum of payroll and non-payroll cost of an NSS F. In addition, if there does not exist an NSS that is included in all required classes, then combinations of NSSs that cover the classes can be considered taking into account cumulative rating-cost ratio for each appropriate combination:

$$\sum_j \frac{R_{F^j}}{C_{F^j}} \to max$$

where $j \in \{j_1, j_2, \dots, j_k\}$, $\{F^{j_1}, F^{j_2}, \dots, F^{j_k}\}$ is the set of NSSs in the combination.

4 A Policy Algebra

All NSSs inside any equivalence class produce equal outputs while implementing policies. However, in the general case their policies from the syntax point of view are different. As can be seen from the above examples, Check Point does not use CLI and it is not possible to compare its policy with the analogue in Cisco from the syntax point of view; however, the policies have equal semantics. Thus, in the general case there are multiple policies that describe the same simple NSS inside an equivalence class. In order to reduce redundancy of policies for NSSs the set of simple policies P_s needs to be substituted for a set of simple unified policies P_u and $|P_u|$ must be minimal. Consequently, only one generalized policy needs to be assigned for every single equivalence class. In other words, if the set $\mathcal{F}_s$ consists of N classes then it is required to have N generalized policies in order to specify policies for all NSSs.

Let $\Omega = \{S, \Phi, \sigma\}$ be a many-sorted signature, where $S = \{s_t, s_m, s_1, s_2, \dots, s_l\}$ is a set of sorts while s_t and s_m are sorts of network traffic and messages respectively, $\Phi = \{+, \varphi_1, \dots, \varphi_N\}$ is a set of functional symbols, $\sigma(\varphi_i) =< s_t, s_{i1}, \dots, s_{ik}, s_m, s_t >$ is a sort function that defines sorts of arguments $s_t, s_{i1}, s_{i2}, \dots, s_{ik}$ and sorts of values s_m, s_t for every functional symbol $\varphi_i \in \Phi$. In addition, suppose that $\sigma(+) =< s_t, s_t, s_t >$, i.e., function "+" has two arguments of the sort of network traffic and a value of the same sort (two examples of the function are shown in Fig. 4).

Let $V = V_{s_t} \cup V_{s_m} \cup V_{s_1} \cup \dots \cup V_{s_l}$ be a set of variables, where V_{s_i} is a set of variables of a certain sort $s_i \in S$. Suppose that Θ is the set of Ω-terms that is recursively defined as follows:

- any variable V_{s_i} is a Ω-term θ_{s_i} of the sort s_i;
- any finite expression $\varphi_i(\theta_{s_t}, \theta_{s_{i1}}, \theta_{s_{i2}}, \dots, \theta_{s_{ik}}, \theta_{s_m}, \theta_{s_t})$ such that $\sigma(\varphi_i) =< s_t, s_{i1}, s_{i2}, \dots, s_{ik}, s_m, s_t >$ is a Ω-term of the sort s_t, where $\theta_{s_t}, \theta_{s_m}$ and $\theta_{s_{ij}}$ are any variables of corresponding sorts s_t, s_m and s_{ij}.

Suppose that any $s_i \in S$ is a name of a certain parameter of generalized policy, any $\varphi_i \in \Phi$ is a designation of a certain equivalence class of NSSs, then $\varphi_i(\theta_{s_t}, \theta_{s_{i1}}, \theta_{s_{i2}}, \dots, \theta_{s_{ik}}, \theta_{s_m}, \theta_{s_t})$ is a representation of the output function of an NSS from equivalence class designated by φ_i that includes representation of respective simple policy. By construction, Θ includes representations of all generalized policies for simple and composite NSSs. For instance, the NSS shown in Fig. 4 can be represented as follows:

$$\begin{cases} F_{fw}(t, p_1, s) + F_{fw}(t, p_2, s) + F_{fw}(t, p_3, s) = t' \\ F_{nat}(t', nat_1, s) + F_{nat}(t', nat_2, s) + F_{nat}(t', default, s) = t'' \end{cases}$$

where F_{fw} and F_{nat} are designations of equivalence classes; p_i, nat_j, $default$, and s are collections of parameters of respective policies and states (parameters of policies consist of input, output and state vectors). Note that notations $+(\theta_{s_t}, \theta_{s_t})$ and $\theta_{s_t} + \theta_{s_t}$ are equivalent.

Let (A, I) be a many-sorted algebra, where A is a carrier set of the algebra and I is an interpretation of the signature Ω. For every sort $s_i \in S$ the interpretation associates a subset $A_{s_i} \subseteq A$ and for every functional symbol $\varphi_i \in \Phi$ it associates the function $f_i: A_{s_t} \times A_{s_{i1}} \times A_{s_{i2}} \times A_{s_{ik}} \to A_{s_m} \times A_{s_t}$ that defines the output function of an NSS of the respective equivalence class F_i. Thus, the algebra models all definite policies for simple NSSs of each equivalence class as well as definite policies for composite NSSs.

In order to translate the policies into the native policy formats of concrete NSSs' platforms described many-sorted algebra can be represented as a formal language with a generative grammar $G = (\mathcal{T}, \mathcal{N}, \mathcal{S}, R)$, where $\mathcal{T}$ is a terminal vocabulary that reflects carrier set A, $\mathcal{N}$ is a non-terminal vocabulary that includes terms, $\mathcal{S} \in \mathcal{N}$ is a starting non-terminal symbol of every policy, and R is a set productions. Representation of the policies in the form of a formal language allows application of the existing parsing algorithms.

5 Conclusion

The approach to policy modeling presented in the paper is based on finite automaton model of an NSS. Decomposition of an NSS to simple NSSs and their classification facilitates composition of policies and selection of the most effective NSSs aligned with them. Policies are considered independently on platforms of NSSs and can be specified identically for equivalent NSSs. Application of translation methods for unified policies allows implementation of policies in concrete NSSs platforms.

The future challenge for the approach is development of derivation method from RBAC policies into described NSSs policies and building of conflict resolution methods. Construction of an algorithm for classification of NSSs is also a future work.

References

1. 2014 Cyberthreat Defense Report North America & Europe, CyberEdge Group (2014)
2. The State of Network Security 2013: Attitudes and Opinions, AlgoSec (2013)
3. Chappell, M.J., D'Arcy, J., Striegel, A.: An Analysis of Firewall Rulebase (Mis)Management Practices, Journal of Information System Security Association (ISSA) 7, 12–18 (2009)
4. Wool, A.: Trends in Firewall Configuration Errors: Measuring the Holes in Swiss Cheese. IEEE Internet Computing 14 (4), 58–65 (2010)
5. Rees, J., Bandyopadhyay, S., Stafford, E.H.: PFIRES: a Policy Framework for Information Security. Communications of the ACM 46 (7), 101-106 (2003)
6. Wahsheh, L.A., Alves-Foss, J.: Security Policy Development: Towards a Life-Cycle and Logic-Based Verification Model. American Journal of Applied Sciences 5 (9), 1117–1126 (2008)
7. Knapp, K.J., Morris, R.F. Jr., Marshall, T.E., Byrd, T.A.: Information security policy: An Organizational-level Process Model. Computers & Security 28 (7), 493–508 (2009)
8. Tuyikeze, T., Potts, D.: An Information Security Policy Development Life Cycle. In: South African Information Security Multi-Conference, pp. 165–176. (2010)
9. Labored, R., Kamel, M., Barrera, F., Benzekri, A.: Implementation of a Formal Security Policy Refinement Process in WBEM Architecture. In: 4th Latin American Network Operations and Management Symposium, pp. 65–76. (2005)
10. Cuppens, F., Cuppens-Boulahia, N., Sans, T., Miège, A.: A Formal Approach to Specify and Deploy a Network Security Policy. In: 2nd Workshop on Formal Aspects in Security and Trust, pp. 203–218. (2004)
11. Preda, S., Cuppens-Boulahia, N., Cuppens, F., Garcia-Alfaro, J., Touting, L.: Model-Driven Security Policy Deployment: Property Oriented Approach. In: 2nd International Symposium on Engineering Secure Software and Systems, pp. 123–139. (2010)
12. Preda, S., Cuppens-Boulahia, N., Cuppens, F., Touting, L.: Architecture-Aware Adaptive Deployment of Contextual Security Policies. In: 5th International Conference on Availability, Reliability and Security, pp. 87–95. (2010)
13. Garcia-Alfaro, J., Cuppens, F., Cuppens-Boulahia, N., Preda, S.: MIRAGE: A Management Tool for the Analysis and Deployment of Network Security Policies. In: 5th international Workshop on Data Privacy Management and 3rd International Conference on Autonomous Spontaneous Security, pp. 203–215. (2010)
14. Rahman, M. A., Al-Shaer, E.: A Formal Framework for Network Security Design Synthesis. In: 33rd International Conference on Distributed Computing Systems, pp. 560–570. (2013)