Spécification Method for Analyzing Fine Grained Network Security Mechanism Configurations

Hicham El Khoury, Romain Laborde, François Barrère, Abdelmalek Benzekri, Chamoun Maroun

A Specification Method for Analyzing Fine Grained Network Security Mechanism Configurations

El Khoury Hicham, Laborde Romain,
Barrère François, Benzekri Abdelmalek
IRIT University Paul Sabatier
Toulouse, France
hkhoury@ul.edu.lb, Romain.Laborde@irit.fr,
Francois.Barrere@irit.fr, Abdelmalek.Benzekri@irit.fr

Chamoun Maroun
Saint Joseph University
Beirut, Lebanon
maroun.chamoun@usj.edu.lb

***Abstract*—Quick evolution, heterogeneity, interdependence between equipment, and many other factors induce high complexity to network security analysis. Although several approaches have proposed different analysis tools, achieving this task requires experienced and proficient security administrators who can handle all these parameters. The challenge is not to propose a temporary solution but to offer a building block for this large domain, though no approach can be optimal for all tasks. In previous papers, we have proposed a novel formal model of equipment configuration built on data flow attribute-based approach to detect network security conflicts. In this paper, we extend the previous proposed model in order to make it more generic by proving it can handle microscopic analysis. We define a formal analysis method for network security mechanisms. Therefore, we specify our approach in Colored Petri Networks to automate the conflicts analysis and test it on a fine-grained firewall scenario.**

Keywords—security; conflict detection; security configurations; formal specification; Colored Petri Nets.

I. INTRODUCTION

Basically, configuring network equipment often consists of rules referring to configuration items. All these rules are jointly responsible for the implementation of a behavior in terms of network (security) policy and must guarantee the administrator's (security) objectives.

However, each configuration rule also affects the global network security. If a rule is poorly defined, the global security might be compromised (principle of the weakest link in the security chain). These configuration rules follow a syntax and an order of configuration that are specific to each type of equipment. When these configurations of network devices are inconsistent, they may lead to abnormal or unexpected behavior. This results in inconsistency problem.

To ensure the compliance of network security configurations to a security policy, two approaches are generally used for protecting data: (1) The Top/Down approach is followed by network management practitioners and consists in using different abstraction levels of management information that help administrators refining configuration from objectives. (2) Whereas the Bottom/Up approach consists in analyzing existing configurations on security devices and deducing the correctness and the consistency of these configurations on the network equipment.

However, experience shows that these two approaches are often hard tasks.

In previous papers [17, 18], we have presented data flow as a sequence of logical elements to match physical data flows and which was a sequence of bytes grouped according to the specifications of the network protocols. The security mechanisms were suggested as transformation functions handling data flows only. We have proven that constraints applied to data flow and security mechanisms can point out conflicts that may occur between heterogeneous mechanisms. In [19], we have completed this work improving the data flow model and have proposed a formal generic attribute-based model for network security mechanisms representation including generic formal configurations. We have defined generic atomic commands, which allow one to build transformation functions by combining them. The importance of a configuration model allows controlling the behavior of transformation functions. Finally, we have tested our approach through various mechanisms such as IPsec, FW and NAPT scenarios.

In this paper, we made a good progress in extending the proposed model so that it becomes not only handle macroscopic models as shown in previous papers (dealing with technologies as black boxes) but also microscopic models (dealing with each mechanism – functionality – provided by one technology).

The rest of this article is organized as follows. Section 2 presents the related works. Section 3 introduces our formal model. Section 4 describes our model specified in Colored Petri Nets. Section 5, illustrates our model with a concrete example based on iptables technology. Finally, Section 6 concludes and exhibits our future work.

II. RELATED WORKS

A variety of approaches have been proposed in the domain of policy conflict analysis and of detecting misconfigurations. For example, firewall modeling, design method and conflict analysis were targeted by [1], [5], [12], [14] and [16]. This classification had been improved and IDS were introduced in [2]. Also, there was a considerable amount of work on detecting misconfiguration in IPSec tunnels, firewalls and IDS ([3], [7] and [15]). These models represent the reality faithfully. But, they are closely attached to a limited set of technologies; therefore, it is difficult to adapt them to other technologies.

[4] has proposed a formal approach to determine if a network configuration including firewalls and IPSec gateways are compliant with the security objectives. The formalization is limited to some information contained in the

IP header while our model represents all the attributes of a network packet. Independency from technologies has been considered by [6]. However, the level of abstraction defined for specification creates difficulties when abstract specification has to be transformed into real configuration.

III. Modeling security based on data flow

We present in this section the foundation of our formal framework for modeling security devices based on data flow. First, we briefly introduce our model of data flow and data flow treatment that were published in [17, 18]. Then we define our model of configuration of devices that improves the one presented in [19].

A. A Formal Data Flow-Oriented Model

In the basic model which was published in [18], a data flow is a contiguous set of bytes of variable size conveyed over a network. We had defined our core entities by:

- $\boldsymbol{\mathcal{A}}$ is the set of possible attributes. An attribute $a \in \mathcal{A}$, represents a couple <name, value>,
- $\boldsymbol{\mathcal{P}}$ is the set of protocols, i.e., the set of logical blocks. The attributes set is defined on the Power-set of $\mathcal{A}$, i. e. , attributes $\in \mathbb{P}(\mathcal{A})$,
- $\boldsymbol{\mathcal{E}} = \mathcal{P}^{\mathbb{N}}$, is the set of finite sequences over $\mathcal{P}$. This set represents all the possible encapsulation chains of protocols,
- $\boldsymbol{\mathcal{S}}$ is the set of security algorithms addressing the encapsulation chain of protocols,
- $\boldsymbol{\mathcal{L}} = \{\text{all, val}\}$ represents how an attribute has been encrypted.

The history of actions of authentication and confidentiality performed on a data flow is maintained by two sets AUTHN and CONF (definition 1).

Definition 1: *Formal definition of data flows*

Based on above the definitions, we present the set of data flows as: $\boldsymbol{\mathcal{F}} \subseteq \boldsymbol{\mathcal{E}} \times \mathbf{AUTHN} \times \mathbf{CONF}$ such that:

- $\mathcal{E}$ is the encapsulation chain of protocols,
- $\text{AUTHN} \subseteq (\mathcal{A} \times \mathcal{P} \times \mathcal{A} \times \mathcal{P} \times \mathcal{S})$, represents the attributes of the data flow that have been authenticated.
- $\text{CONF} \subseteq \text{BAG}(\mathcal{A} \times \mathcal{P} \times \mathcal{S} \times \mathcal{L})$, represents the attributes of the data flow that have been encrypted.

Definition 2: *Basic commands*

A basic command represents the most basic treatment that can be applied on data flows. According to our data flow model, we propose nine basic commands [19]: get/add/delete protocol, get/modify attribute, add/delete authentication, add/delete confidentiality which was modeled in CPN-ML. Any treatment on data flows performed by a security mechanism will be identified as a specific combination built from these basic commands which we will call action. Case studies and complete model implementation details can be found in [19].

B. Abstract attribute-based mechanism model

We supplement data flow model with an abstract model of device configuration using an attribute-based approach. We improve the initial stage of configuration model introduced in [19] to represent and configure a security mechanism in a generic way.

A treatment on a data flow is performed by a specific mechanism with a specific configuration. A specific mechanism has a specific capability that represents what the mechanism can do. For example, a firewall can filter packets by analyzing IP addresses, ports, etc. The second component of a specific mechanism is its configuration. The configuration defines a specific behavior based on the capability of the mechanism. For example, a configuration of a firewall can be "if IP source address equals 1.2.3.4 and TCP source port is less than 1024 then deny". This configuration requires the firewall to be able to (1) retrieve the IP source address and the TCP source port in the packet, (2) apply functions "IP address is equal to" and "port is less than", and (3) apply action "deny". As consequence we define a mechanism M as a specific capability CAPABILITYM and a specific configuration CONFIGURATIONM.

$$\mathbf{M} = < \mathbf{CAPABILITY_M}, \mathbf{CONFIGURATION_M} >$$

1) Formal definition of the capability of a mechanism

The following definitions are the terminology accredited to represent the attribute-based mechanism model. Let A_M^f denote the set of attributes of a data flow f that can be fetched by a special mechanism M, and A_M^{ctx} the set of context attributes found in a rule of M. We call context attributes, attributes used in the configuration rule that are not contained in data flows (e.g. –o eth0 or stateful information in iptables rule). Each attribute is required to have a type that belongs to Σ_M, the set of non-empty types recognized by M. (e.g. @ips is an attribute representing the source IP Address of an IP packet).

We denote:

- $\text{Type}_M(a_i)$ the type of the attribute a_i where $a_i \in A_M^f$. $\text{Type}_M(a_i) \in \Sigma_M$. Example: IP_Address = Type_M(@ips) and STRING = Type_M(protoname).
- $\text{Value}(a_i)$ the value of the attribute a_i. Example: Value(@ips) = 10.2.1.11
- $\text{Values}(a_i)$ the set of values for the attribute a_i. Example: Values(@ips) = 10.2.0.0/16 (Range of IP Addresses) and Values(ports) = [1..1024] (Public port numbers).

We denote by EXPR_M the set of expressions provided by a given M. The type of an expression $e_M \in \text{EXPR}_M$ depends on its functionality, i.e., the type of the results obtained when evaluating e_M. The set of all attributes in an expression e_M is defined as $\text{Attr}(e_M)$.

Definition 3: *Formal definition of* $\boldsymbol{CAPABILITY_M}$

Capability of mechanism M is represented by $\text{CAPABILITY}_M = < \Sigma_M, A_M, \text{EXPR}_M^A >$ Where:

1) $\boldsymbol{\Sigma_M}$ is a non-empty finite set of types;
2) $\boldsymbol{A_M = A_M^f \cup A_M^{ctx}}$ | A_M={a_i | 1 ≤ i ≤ k where $\mathbf{k} \in \mathbb{N}$ and $\text{Type}(a_i) \in \Sigma_M$ }
3) $\boldsymbol{EXPR_M^A}$ is a finite set of expressions.

2) Formal definition of the configuration of a mechanism

Based on these precedents capabilities, we will define the elements of configuration as following:

Definition 4: *Formal definition of* $\boldsymbol{CONFIGURATION_M}$

Configuration of M is a list of rules $RULE_M^A$ (definition 5) and a conflict resolution algorithm 'CRA' (definition 8) where both are based on one or more elements of CAPABILITYM (definition 3):

$$CONFIGURATION_M \in RULES_M \times CRA_M$$

Definition 5: *Formal definition of* $\boldsymbol{RULE_M^A}$

A rule of M consists of a set of constraints on A_M (a set of k fetched attributes), together with an action, $ACTION_M$, from the set of all possible actions based on basic commands (definition 2).

$$RULE_M^A \subseteq CONDITION_M^A \times ACTION_M^A$$

Definition 6: *Formal Definition of* $\boldsymbol{CONDITION_M^A}$

$CONDITION_M^A$ is the set of Boolean expression on A_M that must be satisfied for the action to be triggered, the condition can be represented in the conjunctive normal form $[cond_M] = \{cond_{a_1}, cond_{a_2}, \dots, cond_{a_i} | \, i \in \mathbb{N}\} = \wedge_{i \in N}[cond_{a_i}]$.

$CONDITION_M^A \subseteq EXPR_M^A | \, \forall \, cond_M \in CONDITION_M^A$
where Type($cond_M$)=Bool and $\forall \, e \in Attr(condM), e \in \Sigma_M$

Definition 7: *Formal Definition of* $\boldsymbol{ACTION_M^A}$

$ACTION_M^A$ is the set of action expressions on A_M of type data flow (definition 1) as follow:

$ACTION_M^A \subseteq EXPR_M^A | \, \forall \, action \in ACTION_M^A$
and $Type(action) \in \Sigma_M$

$ACTION_M^A$ is required to be a set of needed and necessary expressions (definition 2) that affect a data flow.

Definition 8: *Conflict Resolution Algorithm* $\boldsymbol{(CRA_M)}$

Conflict may occur with any set of rules where at least two matching rules in $RULE_M^A$ have different actions: Allow, Deny, Protect, and so on. Mechanisms include one or more conflict resolution algorithms to cope with this situation. Examples of such algorithms are deny-takes-precedence, first-match-takes-precedence, more-specific-takes-precedence, etc.

In order to provide a unified way for representing conflict resolution algorithms, we reuse the work of Chinaei et al. [10].

IV. Specification in Colored Petri Nets (CPN)

In order to facilitate the analysis detection task, we have specified our formalism in hierarchical colored Petri nets; this formal language being adapted to our issues [6] and featured with tools (CPN tools [13]) to validate our formal methodology.

```
// Definition of the list of attributes
color ATTRIBUTE = record name:STRING * value:STRING
color ATTList = list ATTRIBUTE;
// Definition of encapsulation chain of protocols
color PROTOCOLID = record name :STRING * id :INT;
color PROTOCOL = record protoid:PROTOCOLID*value:ATTList;
color ENCAPSULATION = list PROTOCOL;
…
color CONF = product ATTRIBUTE*PROTOCOL*SECALGO*LEVEL ;
color CONFLIST = list CONF;
// Definition of data flows
color DATAFLOW = product ENCAPSULATION*AUTHNLIST*CONFLIST;
```

Fig. 1. Definition in CPN-ML of data flow

A. Net structure and declaration of our definition in CPN

We simulate and validate our CPN model with "CPN Tools". The CPN development environment uses an extension of the Meta Language (ML) to formally specify colors of tokens, guards at transitions, and functions on arcs. We have translated the formal definition of data flows (*Fig. 1*) and mechanism (*Fig. 2*) into ML.

```
…
// Definition of rules
color RULE = product CONDITION * ACTION;
color RULES = list RULE;
// Definition of a list of A_M
color AM=product PROTOCOLID*ATTRIBUTE;
…
// Definition of the representation of a mechanism M
color CONFIGURATION = product RULES*CRA;
color M=record capab:CAPABILITY*CONFIG:CONFIGURATION;
```

Fig. 2. Definition in CPN-ML of a mechanism

B. Generic attribute-based mechanism model (GAM)

Our objective is to provide a generic CPN model that could be specialized to represent any security mechanisms. Thanks to hierarchical CPN, this CPN can be considered as a black box representing the basic element for specifying any treatment on data flows.

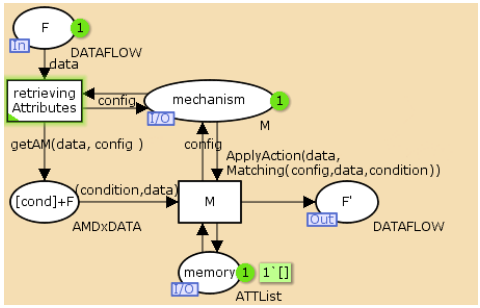

Fig. 3. GAM – a generic attribute-based mechanism model in CPN

Fig. 3, displays our generic attribute-based mechanism model (GAM). It takes as input a DATAFLOW (place F in *Fig. 3* tagged "In") and returns a DATAFLOW as an output (place F' in *Fig. 3* tagged "Out"). The description of the generic mechanism is defined in place "mechanism" which contains the capability and the configuration of the mechanism. Finally, the contextual attributes A_M^{ctx} are stored in place "memory". This memory could be used for representing stateful mechanisms [11]. In addition, place "memory" can be shared by different generic mechanisms.

Informally, the behavior of the GAM is the following:

Input: (DATAFLOW, M)
Output: (DATAFLOW)

Step 1: getAM(DATAFLOW, M)

This function retrieves the perceived attributes A_M from the dataflow (variable condition)

Step 2: Matching(DATAFLOW, M, AMD)

2.1 This function returns all the rules that match the attributes retrieved in the dataflow and the memory.

2.2 If there is at most one matching rule in the list "match", go to step 4.

Step 3: CRA(DATAFLOW, RULES, CONFIGURATION)

At this point, the matching list contains at least two rules (obtained via step2). CRA (definition 8) will be applied in order to rearrange and remove rules that won't be applied and then go to step 4.

Step 4: ApplyAction(DATAFLOW, RULES)

Actions of matching rules are applied. The result is a dataflow which could be empty, unchanged or modified.

V. Case Study

In this section, we present the modeling of the iptables technology [11] to prove that our approach can be used for fine grained specifications. First, we introduce iptables, especially its capability to filter and tag dataflows. Then we present a use case related to filtering and routing. Finally, we specify this example and show how we can discover conflicts.

A. Introduction to iptables

Iptables is an IP Filter which is shipped with Linux kernel [11]. Technically speaking, an IP filter will work on Network layer in TCP/IP stack but actually iptables work on data link and transport layer as well. In a broad sense, iptables consists of tables (Raw, Mangle, NAT, and Filter). Each table has a number of build-in chains (PREROUTING, INPUT, FORWARD, OUTPUT and POSTROUTING) which is further comprised of rules e.g. PREROUTING is used by raw, mangle and nat tables.

In their journey in the TCP/IP stack, packets traverse the different chains. If the packet is coming from the network, it enters in the PREROUTING chain. Then, a routing decision is taken. Depending on the routing decision, the packet is sent to the INPUT chain if the destination is the local host or the FORWARD and the POSTROUTING chains if the destination is a remote host. If a local process sends a packet, it passes through the OUTPUT and PREROUTING chains. For more details refer to the following tutorial [11].

Tables consist of the set of possible actions that iptables can perform (Table Filter is for filtering, table NAT is for DNAT/NAPT). A less known table is MANGLE. A mangle rule allows setting marks on packets. These marks can be used in future processing in their rules' conditions. They identify a packet based on its mark and process it accordingly. The mangle marks exist only within the same Linux system. The mark cannot be transmitted across the network. In addition, the mangle facility is used to modify some fields in the IP header, like type of service TOS (DSCP) and TTL fields.

B. Description of the iptables scenario

Let's consider an administrator who wants to configure Linux router R1 (*Fig. 4*). The administrator has to configure the system for (1) routing SMTP packets to 192.168.2.2, and (2) filtering specific packets.

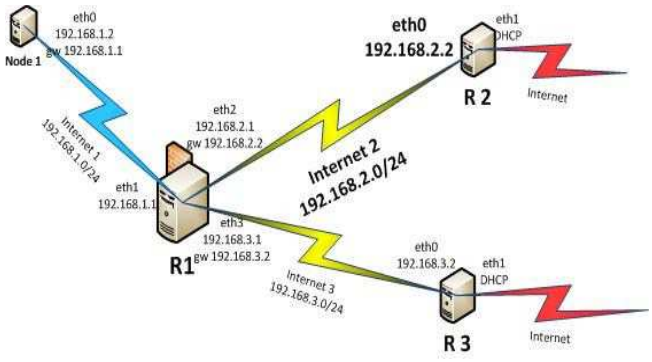

Fig. 4. iptables configuration scenario

1) Routing SMTP packets

The administrator has to configure R1 to route every SMTP packets to 192.168.2.2 (router R2). However, the default route is set to 192.168.3.2 (router R3). Theoretically, routing is based on destination IP addresses only. However, the administrator has read the tutorial in [8] that explains how to use iptables to route packets based on ports. The idea is to use the mangle table to mark SMTP packets and to use this mark for routing them.

Following this technique, the administrator creates a rule in the PREROUTING chain to set '1' in the mark for SMTP packets using the command "--set-xmark" that adds a specific value to the current mark:

 # iptables -t mangle -A PREROUTING -p tcp --dport 25 -j MARK --set-xmark 0×1/0×0

Since the packets are marked with a '1', the following instruction in the routing policy database aims to let outgoing mail be sent via router R2:

 # echo 201 mail.out >> /etc/iproute2/rt_tables

 # ip rule add fwmark 0×1 table mail.out

 # /sbin/ip route add default via 196.168.2.2 dev eth2 table mail.out

1) Filtering packets:

The administrator wants to filter packets addressed to 192.168.2.0/24 with destination port less than 1024. He reads a tutorial [9] that explains how to use table mangle as a good way "to use groups when writing rules, which can simplify things if you've got a potentially large rulebase".

Following the example provided in [9], the administrator has grouped rules in the FORWARD chain intents to filter it. The setup is to mark with a '1' the packet addressed to the 192.168.2.0/24 network and with a '2' the ones addressed to port between 0 and 1024. If a packet comes out with a mark equals '3', it will be dropped. This allows splitting rules in groups related to destination IP addresses and destination ports.

 # iptables –t mangle -A FORWARD –d 192.168.2.0/24 -j MARK –set-xmark 0×1/0×0

 # iptables –t mangle -A FORWARD -p tcp –dport :1024 -j MARK –set-xmark 0×2/0×0

 # iptables -A FORWARD -m mark –mark 0×3 –j DROP

2) The problem

The administrator discovers that a packet for 192.168.2.2 with destination port equals to 25 is forwarded instead of being dropped. And when destination IP address is 192.168.3.2 with destination port 25, the packet is dropped instead of being routed to R2.

C. CPN iptables Scenario

We propose to analyze this scenario with our approach. The interaction between mechanisms in the iptables technology is implemented in CPN tools. Each table in the iptables chains is modeled using a GAM specialized with specific capabilities and configuration corresponding to prerouting mangle, routing, forward mangle and forward filter (*Fig. 5*). Different approaches could be used to specify the mangle mark that is set to a packet. We have decided to represent it in the dataflow. Thus, the first mangle GAM

transforms a dataflow *f =(< ...,ip1,tcp1,... >,{},{})* into {},{}). Another approach could have been to represent it in a shared memory, i.e., the first mangle GAM does not modify the dataflow, but add in the shared memory the mark that can be reused by other GAM. We prefer to use the first approach because the mark is a packet metadata in the Linux kernel.

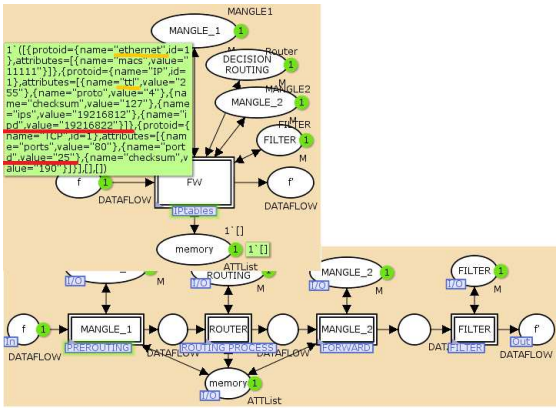

Fig. 5. Navigating through marking menus

Analysis of conflicts can be done by following data flows in each GAM (Mangle_1, router, Mangle_2 or Filter). When injecting a dataflow where destination IP address is equal to 192.168.2.2 and the destination port equals to 25, value of attribute "*fwmark*" of protocol "*mangle-mark*" is equal to 4 in the GAM Filter (Mangle_1 sets mark to '1' and Mangle_2 adds '3' to the mark). As consequence, the packet is forwarded (*Fig. 6*).

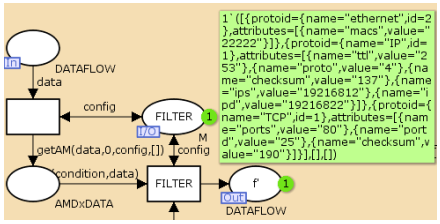

Fig. 6. Accepted data flow by GAM Filter

When injecting a dataflow where destination IP address is equal to 192.168.3.2 and the destination port equals to 25, value of attribute "*fwmark*" of protocol "*mangle-mark*" is equal to 3 in GAM Filter (Mangle_1 sets mark to '1' and Mangle_2 adds '2' to the mark). Thus, the packet is dropped by the GAM filter (*Fig. 7*).

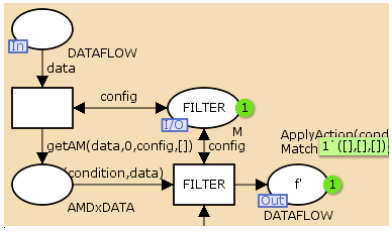

Fig. 7. Dropped data flow by GAM Filter

VI. Conclusion

In this paper, we have presented a formal data flow-oriented approach for specifying security mechanisms. This work improves our previous results by (1) enhancing the model of mechanisms and their configurations, and (2) dataflow *f' =(<(mangle-mark,{<fwmark,1>}),..,ip1,tcp1,..>*, proving our approach can specify complex and fine grained security scenarios. In addition, our model of mechanism being generic and can be specialized without being modified. The whole approach has been represented in CPN for facilitating the use of this model.

Analysis of security conflicts is currently done by simulation only. Our future work will try to help/automate analysis by allowing analyzers to set properties in temporal logic that will be automatically checked by a tool. For example, looking at dataflow token with certain characteristics in some place at some point. Our aim is to provide a model-checking based tool such as [6].

References

[1] E. Al-Shaer, H. Hamed, "Discovery of Policy Anomalies in Distributed Firewalls", in IEEE INFOCOM, 2004.

[2] J. Alfaro, N. Cuppens, F. Cuppens, "Complete analysis of configuration rules to guarantee reliable network security policies", in International Journal of Information Security, 7(2), 2008.

[3] Z. Fu, F. Wu, H. Huang, K. Loh, F. Gong, I. Baldine, C. Xu, "IPSec/VPN Security Policy: Correctness, Conflict Detection and Resolution", in IEEE POLICY, 2001.

[4] J. Guttman, A. Herzog, "Rigorous automated network security management", in International Journal of Information Security, 4(3), 2005.

[5] H. Hamed, E. Al-Shaer, "Taxonomy of Conflicts in Network Security Policies", in IEEE INFOCOM, 2006.

[6] R. Laborde, M. Kamel, F. Barrère, and A. Benzekri, "Implementation of a Formal Security Policy Refinement Process in WBEM Architecture", Journal of Network and Systems Management, 15(2), 2007.

[7] S. Preda, "Reliable context aware security policy deployment with applications to IPv6 environments", PhD thesis, Télécom Bretagne, 2010.

[8] Tutorial-1, www.lartc.org (Accessed in 26 June 2013).

[9] Tutorial-2, www.andys.org.uk/bits/2010/01/27/iptables-fun-with-mark (Accessed in 26 June 2013).

[10] A. Chinaei, H. Chinaei, F. Tompa. "A Unified Conflict Resolution Algorithm" in SDM 2007, LNCS 4721, pp. 1–17, 2007.

[11] iptables, www.frozentux.net/iptables-tutorial (Accessed in 26 June 2013).

[12] E. Al-Shaer, S. AL-Haj, "FlowChecker: Configuration Analysis and Verification of Federated OpenFlow Infrastructures". In SafeConfig 2010.

[13] CPN Tools, www.cpntools.org (Accessed in 26 June 2013).

[14] E. Al-Shaer, W. Marrero, A. El-Atawy,K. ElBadawi, "Network configuration in a box: towards end-to-end verification of network reachability and security". In ICNP 2009.

[15] F. Cuppens, N. cuppens-Boulahia, J. Garcia-Alfaro, T. Moataz, X. Rimasson "Handling Stateful Firewall Anomalies". IFIP SEC 2012.

[16] A. Liu, M. Gouda, "Diverse Firewall Design", IEEE Transactions on parallel and distributed systems, 19(9), 2008.

[17] H. El Khoury, R. Laborde, F. Barrère, A. Benzekri, M. Chamoun, "A Generic Data Flow Security Model" (poster). SafeConfig 2011.

[18] H. El Khoury, R. Laborde, F. Barrère, M. Chamoun, A. Benzekri, "A Formal Data Flow-Oriented Model For Distributed Network Security Conflicts Detection". In ICNS 2012.

[19] H. El Khoury, R. Laborde, M. Chamoun, F. Barrère, A. Benzekri, "A Generic Attribute-Based Model for Network Security Mechanisms Representation and Configuration", in FCST 2012.