

# Formal Implementation of Network Security Policies

Alexandre Lacasse, Mohamed Mejri and Béchir Ktari

Département d'informatique et de génie logiciel

Université Laval, Québec, G1K 7P4, Canada

{Alexandre.Lacasse, Mohamed.Mejri, Bechir.Ktari}@ift.ulaval.ca

**Abstract**—This paper introduces an algebraic approach that aims to enforce a security policy on a given computer network. More precisely, given a network and a security policy, we want to automatically generate the necessary monitors (a single fire-wall or many ones where each of them controls a part of the networks) that force the network to be secure according to the security policy definition. In this approach, the network is formalized as a process  $P$ , the security policy is formally specified as a formula  $\Phi$  and the problem is to find a process  $M$  (monitor) such that  $P \setminus M \models \Phi$ . Once this step is completed, some results about equivalence between processes can be used to distribute the monitor over the network. In other words, the equivalence results aims to break the monitor  $M$  into small slices that will be distributed so that each slice controls only a small part of the network.

## I. INTRODUCTION

Building secure networks (networks that respect some given security policies) is known as hard challenge. The increasing number of network security flaws is a sufficient witness for this fact. This is due, on one hand, to the complexity of network components and, in the other hand, to the absence of well established and efficient formal methods that resolve this problem. Most of used techniques are *ad hoc* and far from satisfying most of institutions and specially the critical ones (financial institution, military institution, etc.).

Lately, many attempts of proposing new formal approaches or adapting existing ones to deal with network security problem have been proposed. In [1], for instance, the authors extend the State Transition Analysis Technique (STAT [2]) to make it suitable for detecting Network-Based attacks. Basically, once the network topology and the attacks are defined, the technique allows to automatically generate probes, which are monitors that observe the network in order to detect undesired behaviors.

The method presented in this paper exploits a process algebra as formal language to specify, at an abstract level, a given network (topology together with the behaviors of network components) and a logic to specify network security policies. Once the inputs was specified (network as a process and the security policy as a logical formula), a monitor (a new process that behaviors as a firewall) is extracted from the security policy and combined with the network to restrict its behavior so that it could never execute bad actions. In other words, the main intent of this approach is to automatically add to a network a distributed firewall that enforces a given

security policy.

The remainder of this paper is organized as follows: Section II gives more detail about the different steps involved by our methodology. In section III, we define the syntax and the operational semantics of our process algebra. In addition, we give some algebraic rules that can be used to break a monitor into slices and distribute them over the analyzed network. After this, we briefly discuss, in Section IV, the logic used to specify security policies. Then, we illustrate, in Section V, the whole approach by a concrete example. Finally, in Section VI, some concluding remarks on this work and future work are ultimately sketched as a conclusion.

## II. METHODOLOGY

Hereafter, we explain the methodology of our approach for solving the problem of formal enforcement of network security policies. This methodology involves four steps as following:

- 1) The network is specified, at an abstract level, by a process  $P$  in the process algebra defined in Section III.
- 2) The security policy is specified by a formula  $\Phi$  in the logic defined in section IV.
- 3) From  $\Phi$ , we generate a monitor  $M$  so that  $P \setminus M \models \Phi$ .
- 4) Using some algebraic optimization rules, we distribute  $M$  over  $P$ .

To reach the first step, we need to define a process algebra (syntax + semantics) that is suitable for the formal specification of network components and topologies at an abstract level. The most important feature of such algebra are concurrency and communication. Each component (machine, router, etc.) of the network is abstracted by its communication actions that it performs with the other components in the same network. Furthermore, the process algebra should allow us to easily monitor a processes by another. The monitored process is, in our case, the network and the monitor is generated from the security policy.

For the second step, we need a formal language (e.g. a logic) suitable for security policies that we want to enforce. Notice that security properties that we can enforce are safety properties (bad things should never happen).

Once, we have specified the network as well as the security policy, we need to transform this security policy as a process that will play the role of monitor. More formally, let  $P$  be the process that models the network (i.e, the formal representation

of the network) and  $\Phi$ , the security policy, we need to extract from  $\Phi$  a process  $M$  such that  $P \setminus M \models \Phi$ .

Finally, we want to distribute the monitor over the network. This can be done by establishing some distribution properties of the operator used by the algebra to specify networks and monitors. For instance, under some conditions we can have  $(P_1|P_2)/M \sim (P_1/M)|(P_2/M)$  or  $(P_1|P_2)/(M_1|M_2) \sim (P_1/M_1)|(P_2/M_2)$ . The generated slices of monitors can be seen as a firewalls for some hosts, more powerfully IDS for other hosts, or any another network security components.

### III. NEW PROCESS ALGEBRA

The use process algebras in the area of specification and verification of concurrent communicating systems becomes famous thanks to the Milner's CCS [3], [4], the Hoare's CSP [5], and the Bergstra and Klop's ACP [6]. Many publications show how to use process algebras to formally specify and verify complex systems such as computer networks. For instance, in [7] the authors use process algebra to prove that the CSMA/CD-protocol respects some properties. Other interesting and elegant questions have been addressed using process algebra. Among others, we find the problem of interface equations [8] which can be stated as follows: for a given definition of an equivalence relation (generally denoted by  $\sim$ ) and two processes  $A$  and  $B$ , the problem is to find a new process  $X$  such that  $A|X$  (  $A$  combined to  $X$  using a given operator  $|$ ) is equivalent to  $B$ , i.e:

$$A|X \sim B \quad (1)$$

Notice also that many tools such as FDR [9] that support these kind of analysis, commonly known as model checking, are now available.

In the rest of this section, we define a process algebra inspired from Milner's CCS in which we introduce the notion of monitor. The particularity of this new algebra is that its restriction operator allowing to restrict a process by another process instead of a process by a set of actions as it is the case for CCS. This second process is called monitor. So the equation we want to solve in this paper is the following:

$$P \setminus X \models \Phi \quad (2)$$

which is close to the equation 1 except that we talk about monitoring rather than concurrency. In this equation  $P$  is a formal description of the analyzed protocol at an abstract level,  $X$  is some new components (such firewalls) that we want to add to the network and  $\Phi$  is the security policy. In other words, given a networks ( $P$ ) and a security policy ( $\Phi$ ), the question is what are the components ( $X$ ) that we can add to the network so that the security policy  $\Phi$  will be respected?

Hereafter, we describe the syntax and semantics of our process algebra.

#### A. Atomic actions

In keeping with CCS and CSP, we build a process starting with elementary actions. In our algebra, we first regard an elementary action as a message that a process can send through

a channel. So  $a(m)$  is read : "A message  $m$  can be sent through the channel  $a$ ". A process can receive a message by executing an over-lined action, so it can receive the message  $m$  from the channel  $a$  if and only if it can execute the action  $\bar{a}(m)$ . A third kind of action is introduced to restrict the access to a channel. These actions are over-lined by a tilde and will be use to build the monitors.

We can thus build processes with send-action  $a$ , receive-action  $\bar{a}$  or control-action  $\tilde{a}$ . We will use Greek letter to denote general action, that can be send-action, receive-action, control-action or *communication*. A communication is special action, which will be define later. We note  $c(a)$  a communication on the channel  $a$ . So we note  $\alpha$  any action belonging to the set  $\{a, \bar{a}, \tilde{a}\}$ , in the same way,  $\beta \in \{a, \bar{a}, \tilde{a}, c(a)\}$  and  $\gamma \in \{a, \bar{a}, c(a)\}$ . Notice that if we write  $a$  and a Greek letter in a same proposition, these two symbol denote the same action, but maybe in a different form. In the other cases, when an action is noted by non over-lined Latin letter, it could not be a receive-action or a control-action. Finally, we use the abstract term *message* to denote and exchanged data between processes.

#### B. Syntax

The syntax of our algebra, see table I, is very similar to the CCS's one. We use the same operators " $\stackrel{\text{def}}{=}$ " (definition), " $\cdot$ " (prefix) and " $+$ " (choice) with the same meaning. The operator  $\parallel$  is the parallelism operator, sometimes we will write it  $|$  as in CCS to emphasize the fact that two processes do not communicate together. The restriction operator  $\setminus$  is used to restrict a process by another one (rather than by a set of actions like in CCS). When we have two processes linked by the restriction operator, the left process is called the monitored process and the right process the monitor.

$$P ::= 0 \mid \alpha.P \mid P_1 + P_2 \mid P_1 \parallel P_2 \mid P_1 \setminus P_2 \mid P_1 \stackrel{\text{def}}{=} P_2$$

TABLE I  
SYNTAX

#### C. Operational Semantics

First we give the semantics for the three operators that have the same semantics than the ones of CCS.

$$\begin{aligned} (Pre) & \frac{\square}{\alpha.P \xrightarrow{\alpha} P} \\ (Def) & \frac{P \stackrel{\text{def}}{=} Q \quad Q \xrightarrow{\alpha} Q'}{P \xrightarrow{\alpha} Q'} \\ (Choice) & \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'} \end{aligned}$$

TABLE II  
OPERATIONAL SEMANTICS OF  $\stackrel{\text{def}}{=}$ ,  $\cdot$  AND  $+$ .

The arrow over which we place an action symbolizes the evolution of a process by execution this action. The rule

(*Pre*) states that "after executing the action  $\alpha$ , the process  $\alpha.P$  becomes the process  $P$ ", this evolution is denoted by  $\alpha.P \xrightarrow{\alpha} P$ . The fact that a process  $P$  is not able to execute the action  $a$  is denoted by  $P \not\xrightarrow{a}$ . In other words, it doesn't exist a process  $Q$  and an action  $x$  such that  $P \xrightarrow{x} Q$ . Process 0 is a process which cannot execute any action, so we have  $0 \not\xrightarrow{x}$  for any action  $x$ . Recursive process such that  $C \stackrel{\text{def}}{=} \text{tick}.C$  can only be defined using the  $\stackrel{\text{def}}{=}$  notation.

Notice that the choice and the parallel operators are commutative. Also, when the process 0 is implied in a choice or a communication, we can simplify the whole process by eliminating this 0. So we have the following simplification relation:

$$\begin{aligned} P + Q &\equiv Q + P \\ P \parallel Q &\equiv P \parallel Q \\ P + 0 &\equiv P \\ P \parallel 0 &\equiv P \\ P \setminus 0 &\equiv P \end{aligned}$$

The contribution of the relation  $\equiv$  in the operational semantics is through the following rule:

$$\frac{P \equiv P_1 \quad P_1 \xrightarrow{\alpha} P_2 \quad P_2 \equiv P'}{P \xrightarrow{\alpha} P'}$$

The semantics of parallel composition of processes using the operator  $\parallel$  is as shown in table III. Table IV shows when and how a process restricted by another one can evolve.

(C1)	$\frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\bar{\alpha}} Q'}{P \parallel Q \xrightarrow{c(a)} P' \parallel Q'}$	(Communication)
(C2)	$\frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\bar{\alpha}} Q'}{P \parallel Q \xrightarrow{\bar{\alpha}} P' \parallel Q'}$	(Half Communication)
(C3)	$\frac{P \xrightarrow{\beta} P'}{P \parallel Q \xrightarrow{\beta} P' \parallel Q}$	(Parallellism)

TABLE III  
OPERATIONAL SEMANTICS OF  $\parallel$

- The rule (C1) defines a complete communication between two processes. One process  $P$  executes a send-action and the other process  $Q$  executes a receive-action. It results a new kind of action called communication and denoted by  $c(a)$ .
- The rule (C2) defines a half communication between two processes: one process executes a send-action but the second does not execute a receive-action but only a control-action. The real difference between (C1) and (C2) will be clarified by the semantics of the restriction operator.
- (C3) allows to a process to evolve without participating in a communication.

(M1)	$\frac{P \xrightarrow{\alpha} P' \quad M \xrightarrow{\bar{\alpha}} M'}{P \setminus M \xrightarrow{\alpha} P' \setminus M}$	(Non Monitored Action)
(M2)	$\frac{P \xrightarrow{\alpha} P' \quad M \xrightarrow{\bar{\alpha}} M'}{P \setminus M \xrightarrow{\bar{\alpha}} P' \setminus M'}$	(Intercepted Sending)
(M3)	$\frac{P \xrightarrow{\bar{\alpha}} P' \quad M \xrightarrow{\alpha} M'}{P \setminus M \xrightarrow{\bar{\alpha}} P' \setminus M'} P \not\xrightarrow{c(a)}$	(Intercepted Reception)
(M4)	$\frac{P \xrightarrow{\alpha} P' \quad M \xrightarrow{\alpha} M'}{P \setminus M \xrightarrow{\alpha} P' \setminus M'}$	(Synchronism)

TABLE IV  
OPERATIONAL SEMANTICS OF  $\setminus$

Basically, a monitor is a process that is used to control another one. It can prevent a process to do some actions or break communication between two processes.

- The rule (M1) shows the evolution of process when it is not concerned by the monitor restrictions.
- The rule (M2) defines the evolution of process and monitor when sending action is prohibited. Notice that in this case, the monitored process is not blocked, but the send-action is replaced by a control-action. This capture the fact that the process sends its message that is intercepted by the monitor.
- The rule (M3) prevents a process to receive data from a controlled channel. This rule doesn't only replace the receive-action by a control-action, it blocks the process evolution. Combined with the rule (C2), (M3) shows that when two processes want to communicate with each other and this communication is prohibited by the monitor, the sent message will be intercepted by this monitor, the process that sends the message will evolve and the one that is waiting for the message will be blocked.
- The rule (M4) allows a process and its monitor to progress together if they can execute the same action. When the monitor and the controlled process evolve together, this means that monitor doesn't prohibit the action but it notices this event that can be a trigger to prevent some future actions.

#### D. Examples

Let  $P$  and  $Q$  be two non-restricted processes (*i.e.* processes build without using  $\setminus$ ) such that  $P \xrightarrow{\alpha} P'$ ,  $Q \xrightarrow{\bar{\alpha}} Q'$ . Let  $M$  be a monitor defined as follows:  $M \stackrel{\text{def}}{=} \bar{\alpha}.M$ . The processes  $P$  and  $Q$  can communicate together, we have:

$$P \parallel Q \xrightarrow{c(a)} P' \parallel Q' \quad (C1).$$

The monitor  $M$  restricts communication on the channel  $a$ :

$$\begin{aligned} (P \parallel Q) \setminus M &\not\xrightarrow{c(a)} \\ (P \parallel Q) \setminus M &\xrightarrow{\bar{\alpha}} (P' \parallel Q) \setminus M \quad (M2), (C2). \end{aligned}$$

Similar result is obtained when the monitor  $M$  controls just  $P$  or  $Q$ .

$$\begin{aligned} (P \setminus M) \parallel Q &\not\stackrel{c(a)}{\sim} \\ (P \setminus M) \parallel Q &\stackrel{\tilde{a}}{\sim} (P' \setminus M) \parallel Q \quad (C3), (M2) \\ P \parallel (Q \setminus M) &\not\stackrel{c(a)}{\sim} \\ P \parallel (Q \setminus M) &\stackrel{\tilde{a}}{\sim} P' \parallel (Q \setminus M) \quad (C2), (M3). \end{aligned}$$

The processes given above can also perform other actions, for example, the process  $P \parallel (Q \setminus M)$  can execute  $a$  to become  $P' \parallel (Q \setminus M)$ . We call *communication* an evolution of a process that implies the use of the rule (C1) and *interception* an evolution of a process that implies the use of the rule (M2) or (M3), and (C2). For instance  $P \parallel (Q \setminus M) \stackrel{\tilde{a}}{\sim} P' \parallel (Q \setminus M)$  is an  $a$ -interception, and  $P \parallel Q \stackrel{c(a)}{\sim} P' \parallel Q$  denotes an  $a$ -communication, however  $P \parallel (Q \setminus M) \stackrel{a}{\sim} P' \parallel (Q \setminus M)$  is neither a communication nor an interception.

#### E. Algebraic Optimization Rules

In this last part of section about our process algebra, we introduce some optimization rules used in example V to distribute monitor over a network.

By optimization rule, we mean an equivalence rule between processes that allows us to break or/and distribute a monitor over a network. Hereafter, we give some optimization rules.

- Under some conditions we have:

$$(P \parallel Q) \setminus M \approx (P \setminus M) \parallel Q \approx (P \setminus M) \parallel Q$$

This rule can be applied when the monitor  $M$  controls the communication between the two processes  $P$  and  $Q$  and means that controlling both  $P$  and  $Q$  is equivalent to control  $P$  or  $Q$ .

- A rule  $P \setminus M \approx P$  can be applied when the actions of  $P$  are not controlled by  $M$ .
- A rule  $(P \parallel Q) \setminus (M_a | M_b) \approx (P \setminus M_a) \parallel (Q \setminus M_b)$  can be applied when the actions of  $P$  are not controlled by  $M_b$  and the actions of  $Q$  are not controlled by  $M_a$ .

Notice that those rules represent a very small outline of all those possible and a deep studies of the process algebra operators can lead to a complete set of optimization rules.

## IV. LOGIC

In this section we describe a simple logic we use at present to specify security policies before from which we extract monitors.

#### A. Security policies

Before creating a monitor for a network process, we must define in a formal language the security policies we want to enforce. These policies can specify for instance that some computers in the network should not communicate with Internet or with an other host in the network. They can also consist in more complicated rules defining an illicit operation. So, we have to choose an adequate logic which must be expressive enough to specify the safety properties. We choose a logic that

operates on a trace-based model. The syntax of this logic is illustrated in table V and its semantics is given in table VI.

#### B. Syntax

$$\varphi ::= p \mid p.\varphi \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \alpha X\varphi$$

TABLE V  
SYNTAX

In this syntax,  $p$  is a predicate, in our example we will use two predicates, they are :

- $receive(H, src = @)$
- $send(H, dest = @)$

The first is true if the host  $H$  receive a message with source address equals to  $@$  and the second is true if  $H$  send a message with destination address equals to  $@$ . The point denotes concatenation: a trace satisfies  $p.\varphi$  if its first action satisfies the predicate  $p$  and if its remainder satisfies  $\varphi$ . We have the usual logical operator such as *or* and *negation*. Finally, the operator  $\alpha X\varphi$  is a fixed point operator. A trace satisfies  $\alpha X\varphi$  if it satisfies  $\varphi$  in which we replace  $X$  by  $\alpha X\varphi$ . For instance, if  $\varphi = p.X$ , then we have  $t \models \alpha Xp.X$  if and only if  $t \models p.\alpha Xp.X$ . So the trace  $t = a_1.a_2.\dots$  satisfies  $\alpha Xp$  if each  $a_i$  satisfies  $p$ .

Since  $\alpha X\varphi.X$  is a frequently used formula, we define the short cut operator  $*$  as follows :

$$\varphi^* := \alpha X\varphi.X.$$

We can also express the logical *and* with  $\vee$  and  $\neg$ :

$$\varphi_1 \wedge \varphi_2 := \neg(\neg\varphi_1 \vee \neg\varphi_2).$$

#### C. Semantics

We say that a process  $P$  satisfies a formula  $\varphi$ , written  $P \models \varphi$ , if each execution trace  $t = a_1.a_2.a_3.\dots$  of  $P$  satisfies  $\varphi$ . The satisfaction relation between execution traces and formulas is defined inductively in the table VI. In this table,  $t$  and  $t_1$  are execution traces,  $a$  is an elementary action and  $\nu$  corresponds to a valuation function that allows as to say if an elementary action satisfies a proposition.

$t \models p$	iff	$t = a.t_1$ $\nu(p, a) = true$
$t \models p.\varphi$	iff	$t = a.t_1$ $a \models p$ $t_1 \models \varphi$
$t \models \varphi_1 \vee \varphi_2$	iff	$t \models \varphi_1$ or $t \models \varphi_2$
$t \models \neg\varphi$	iff	$t \not\models \varphi$
$t \models \alpha X\varphi$	iff	$t \models \varphi[\alpha X\varphi/X]$

TABLE VI  
SEMANTICS

It is important to notices that this logic is given to show the basic idea of our work and we are working on the definition

of a more appropriate one for the specification of network security policies.

## V. EXAMPLE

In this section, we will see how to use our process algebra and the logic of section IV to enforce a given security policy in the small network illustrated in figure 1. Although this network is not large enough to represent a realistic problem, this example shows the potential of the use of process algebras.

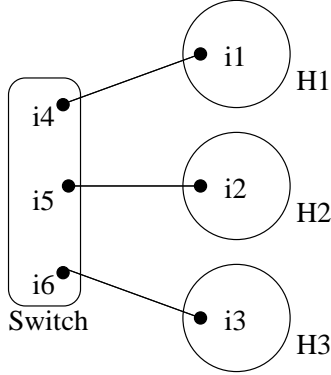


Fig. 1. Three hosts connected via a switch.

First we have to specify this network as a process:

$$\begin{aligned}
 H_i &:= \text{int}_i(@_s, @_d).H_i \\
 &\quad + \overline{\text{int}}_{i+3}(@_s, @_d).H_i \\
 S &:= \sum_{i=1}^3 \overline{\text{int}}_i(@_s, @_d \in \text{Net} \setminus \{@_i\}) \\
 &\quad .\text{int}_{p(@_d)}(@_s, @_d).S + \\
 &\quad \sum_{i=1}^3 \text{int}_i(@_s, @_d \notin \text{Net} \setminus \{@_i\}).S \\
 N &:= S \parallel (H1 \mid H2 \mid H3)
 \end{aligned}$$

Thus an host  $H_i$  can only send a message to the switch or receive a message from it. The switch  $S$  can receive a message from an host and retransmit it to its destination if this one belongs to the network. The function  $p(@)$  appearing in the definition of  $S$  returns the label of the interface communicating with the host having the address  $@$ , for example  $p(@_1) = 4$ . The network  $N$  is composed of a switch that communicates with the other hosts. We then use the operator  $|$  instead of  $\parallel$  with the same meaning to emphasize the fact that hosts cannot communicate together. We can now see the generated trace if  $H_1$  sends a message to  $H_2$  :

$$\begin{array}{ccc}
 N & \xrightarrow{c(\text{int}_1(@_1, @_2))} & N' \\
 N' & \xrightarrow{c(\text{int}_5(@_1, @_2))} & N
 \end{array}$$

The message represented by  $(@_1, @_2)$  is thrown from the interface 1 of  $H_1$  and caught by the switch, what places the network in an intermediate state  $N'$ . Then it passes from the

interface 5 of the switch to  $H_2$  and the network returns to its initial state  $N$ . So  $H_1$  has communicated with  $H_2$  by sending a message to it.

Suppose now we want to implement this security policy :

$H_2$  is not allowed to receive a message with source address equals to  $@_1$  and  $H_1$  is not allowed to send a message with destination address equals to  $@_2$ .

First we have to translate this sentence in logical formula to obtain:

$$\Phi = (\neg \text{receive}(H_2, \text{src} = @_1))^* \wedge (\neg \text{send}(H_1, \text{dest} = @_2))^*$$

The monitor generated from  $\Phi$  must control the critical interfaces  $i_1$  and  $i_5$ . We can rewrite  $\Phi$  using the predicate  $p_1 = \text{int}_1(@, @_2)$  and  $p_2 = \text{int}_5(@_1, @)$  where  $\nu(p_1, a)$  returns true if  $a = c(\text{int}_1(@, @_2))$  and  $\nu(p_2, a)$  returns true if  $a = c(\text{int}_5(@_1, @))$ . So we have to satisfy the formula

$$\Phi' = (\neg c(\text{int}_1(@, @_2))^* \wedge (\neg c(\text{int}_5(@_1, @))^*.$$

The network  $N$  doesn't satisfy  $\Phi'$ , so we restrict it by the following monitor:

$$\begin{aligned}
 M &:= M1 \mid M5 \\
 M1 &:= \widetilde{\text{int}}_1(@, @_2).M1 \\
 M5 &:= \widetilde{\text{int}}_5(@_1, @).M5
 \end{aligned}$$

The monitored network is

$$N_M := S \parallel (H1 \mid H2 \mid H3) \setminus M$$

where the monitor  $M$  can be moved towards  $S$  or broken in two smaller monitors  $M1$  and  $M5$  respectively moved towards  $H1$  and  $H3$ . Thus the two monitored networks  $N_1$  and  $N_2$  defined below models also  $\Phi$ .

$$\begin{aligned}
 N_1 &:= (S \setminus M) \parallel (H1 \mid H2 \mid H3) \\
 N_2 &:= S \parallel ((H1 \setminus M1) \mid (H2 \setminus M5) \mid H3)
 \end{aligned}$$

In the configuration  $N_2$ , the monitors correspond to software security components implemented directly in the monitored hosts, so they are firewalls which block the undesired communications. In the configuration  $N_1$  control is centered on the switch, the monitor knows which messages must be blocked on some ports of the switch.

Finally, we can easily see that it is not sufficient to monitor only  $H2$ . Indeed  $H1$  can send a message with spoofing source address.

$$H1 \xrightarrow{\text{int}_1(@_3, @_2)}$$

This message could not be intercepted without a second monitor supervising  $H1$ .

## VI. CONCLUSION AND FUTURE WORKS

In this paper, we have explained our method to solve the problem of automatically generate a formally proved solution for making secure a network with respect to a given security policy.

First we specify the networks with the process algebra that we have developed. This algebra was inspired from the famous Milner's CCS. Then we formally specify the security policies, which is done using a logic.

After that, we build a monitor as process that is extracted from the logical formula implementing the security policies. These three objects are linked by the relation  $N \setminus M \models \Phi$ . The breaking and the distribution of the monitor  $M$  over the the network  $N$  allows to obtain a more suitable solution to the problem. The final process is the process  $N$  monitored so that it satisfies the formula  $\Phi$ . This final process should correspond to the original network in which we have added some security software components. The proof correctness should ensure that the made safe network respect the security policies.

As a future work, we need to study the efficiency of this approach when applied to real big networks with some complicated security policies. We need also to study more suitable logic and how to automatically generate monitor from a formula. The research must also continue to extract all algebraic optimization rules allowing us to break and diffuse the monitors over a network. The correctness proof remains also an important part of our future works.

We conclude this paper with a last remark concerning our process algebra. This algebra allows us to build monitors which can absorb messages and thus they modify the network behavior. However by changing the rule (C2) and by

combining the rules (M2) and (M3) thus:

$$(C2^*) \quad \frac{P \xrightarrow{\delta} P' \quad Q \xrightarrow{\tilde{a}} Q'}{P \parallel Q \xrightarrow{\tilde{a}} P' \parallel Q'}$$

$$(M2^*) \quad \frac{P \xrightarrow{\delta} M \xrightarrow{\tilde{a}} M'}{P \setminus M \xrightarrow{\tilde{a}} P' \setminus M'}$$

where  $\delta \in \{a, \bar{a}\}$ , we obtain another algebra where the monitor doesn't absorb the messages but only marks them. With this semantics, the actions over-lined by a tilde should correspond to the events we want to keep in a log file. We thus obtain a method to implement passive monitors, what was made in [1] by a different approach. It is an other interesting way to extend the present work.

## REFERENCES

- [1] G. Vigna and R. Kemmerer, "Netstat: A network-based intrusion detection system," *Journal of Computer Security*, vol. 7, no. 1, 1999.
- [2] R. K. K. Iglun and P. Porras, "State transition analysis: A rule-based intrusion detection system," *IEEE Transaction on Software Engineering*, vol. 21, no. 3, 1995.
- [3] R. Milner, "A calculus for communicating systems," *Lecture Notes in Computer Science*, vol. 92, 1980.
- [4] —, "Lectures on a calculus for communicating systems," *Lecture Notes in Computer Science*, vol. 197, 1985.
- [5] C. Hoare, *Communication Sequential Processes (CSP)*. Prentice Hall International, 1985.
- [6] J. A. Bergstra and J. W. Klop, "Algebra of communicating processes with abstraction," *Theoretical Computer Science*, vol. 37, no. 1, pp. 77–121, May 1985.
- [7] J. Parrow, "Verifying a csma/cd-protocol with CCS," *Protocol Specification, Testing and Verification*, 1988.
- [8] —, "Submodule construction as equation solving in CCS," pp. 175–202, 1989.
- [9] F. S. E. Ltd, *Failures Divergence Refinement-User Manual and Tutorial*, 1993, edition: Version 1.3.