Intrusion Detections in Collaborative Organizations by Preserving Privacy

Verma Nischal, François Trousset, Pascal Poncelet, Florent Masseglia

Chapter 1
Intrusion Detections in Collaborative Organizations by Preserving Privacy

Nischal Verma, François Trousset, Pascal Poncelet and Florent Masseglia

Abstract To overcome the problem of attacks on networks, new Intrusion Detection System (IDS) approaches have been proposed in recent years. They consist in identifying signatures of known attacks to compare them to each request and determine whether it is an attack or not. However, these methods are set to default when the attack is unknown from the database of signatures. Usually this problem is solved by calling human expertise to update the database of signatures. However, it is frequent that an attack has already been detected by another organization and it would be useful to be able to benefit from this knowledge to enrich the database of signatures. Unfortunately this information is not so easy to obtain. In fact organizations do not necessarily want to spread the information that they have already faced this type of attack. In this paper we propose a new approach to intrusion detection in a collaborative environment but by preserving the privacy of the collaborative organizations. Our approach works for any signature that may be written as a regular expression insuring that no information is disclosed on the content of the sites.

Nischal Verma
Indian Institute of Technology Guwahati, Assam, India
e-mail: nischaliit@gmail.com

Fran ccois Trousset
LGI2P - Ecole des Mines d'Alès, Parc Scientifique G. Besse, 30035 Nîmes, France
e-mail: francois.trousset@mines-ales.fr

Pascal Poncelet
LIRMM UMR CNRS 5506, 161 Rue Ada, 34392 Montpellier Cedex 5, France
e-mail: poncelet@lirmm.fr

Florent Masseglia
INRIA Sophia Antipolis, route des Lucioles - BP 93, 06902 Sophia Antipolis, France
e-mail: florent.masseglia@sophia.inria.fr

1.1 Introduction

The fast growing computational Grid environments has increased risk of attack and intrusion. Thus misuse detection has become a real concern for companies and organizations. Whereas earlier attacks focused on Web servers which were often misconfigured or poorly maintained, the most recent ones take advantage of Security service and Web application weaknesses which become more vulnerable [6, 5, 3]. To overcome this problem, new approaches called Intrusion Detection Systems (IDS) have been developed. Installed on networks, they aim to analyze traffic requests and detect malicious behavior (eg Prelude-IDS, Snort). They can be classified into two broad categories(*e.g.* [11, 12]): the *Anomaly Detection Systems* which attempt to detect attacks and the *Abuse Detection Systems* which detects unknown comportement so called *abuse* from a specification of allowed ones. Within this paper, we particulary focus on anomaly detection. Their principle mostly consist of matching new requests which signatures of attacks represented as regular expressions. For example, an attack which seeks to recover the password file of a system (*e.g. abc/../de/../../../fg/../etc/passwd*) may be detected by matching with the following regular expression *(/[ˆ ./]*/..)*/etc/passwd*. These signatures are often obtained by using machine learning techniques or from specialized sites (*e.g.* OSVDB [2])

Even if these systems are widely used today, the essential problem is that they do not know how to manage attacks outside their own signature database. When a request is not recognized by the IDS, an alarm is triggered to require external valuation.

Recently approaches called Collaborative Intrusion Detection Systems (CIDS) (*e.g.* [1, 15, 8, 10, 14]) have been proposed. In comparison with isolated IDS, CIDS significantly improve time and efficiency of misuse detections by sharing information on attacks between distributed IDS from one or more organizations. The main principle of these approaches is to exchange information using peer to peer links. However the exchanged information are mostly limited to IP addresses of requests (*e.g.* [1, 8, 10]) and consider that data can be freely exchanged among the peers. The last constraint is very strong: companies, for reasons of confidentiality, do not want to spread out that they were attacked and therefore are unwilling to give any information on it. In this article we propose a secure collaborative detection approach, called SREXM (*Secure Regular Expression Mapping*), which ensures that private data will not be disclosed. Via our approach, regular expressions from the various collaborative sites can be matched without disclosing any information from the local IDS to the outside. Collaborative sites are free to work with signatures of attacks or non-attacks and may give information on the type of intrusion detected. Thus, when new request is checked, the response will be one of: it is an attack (with its type if available), it is a non-attack, or undefined (if none of the IDS data leads to a positive or negative conclusion). To our knowledge, very few studies are concerned with this topic of security in such collaborative environment. The only works [13, 10] consider both collaborative and security aspects. In its context, security mainly concerns information on IP addresses and ports. It uses Bloom's filters to manage data exchanges. Our problem is different in that, we want to exchange

data, *i.e.* more complex than IP addresses and ports. In fact we wants to exchange and parse regular expressions on the full request.

The article is organized as follows. In section 1.2, we present the problem. An overview of our approach is given in section 1.3. The various algorithms are described in section 1.4. Finally section 1.5 concludes and presents various perspectives.

1.2 Problem statement

DB is a database such as $DB = DB_1 \bigcup DB_2 \ldots \bigcup DB_D$. Each database DB_i is equivalent to a tuple $< id, S_{exp} >$ where id is the identifier of the database and S_{exp} is a set of regular expressions. Each regular expression $exp_i \in S_{exp}$ is expressed as a deterministic automaton (*e.g.* [7]) by the tuple $a_{exp_i} =< State, Trans, Init, Final >$. In this tuple a_{exp_i}, $State$ is the set of states of the automaton, $Init$ is the initial state, $Final$ is the set of final states and $Trans$ is the set of transitions. Each transition is a quadruplet $(S_{Initial}, Condition, S_{Final}, Length)$ meaning that if the automaton is in state $S_{Initial}$ and that $Condition$ is checked then automaton current state changes to S_{Final} and move the current position in the filtered string of the amount given by $Length$. In our approach, we also associate a value to each final state. This value is used to specify whether or not it is an attack (boolean 0 or 1), but may also provide the type of the attack (integer).

Example 1 *Consider the following regular expression: (/[^/]*/..)*/etc/passwd. Its associated automaton is described in Figure 1.1. The left table is the matrix of transitions where Conditions are indexes in the second table which contains the effective patterns to be matched with the request string. For example, to move from state S_6 to final state F, we have to check that the request string at current position contains the word "passwd".*

Definition 1 Given a database $DB = DB_1 \bigcup DB_2 \ldots \bigcup DB_D$ and a request string R, the securized approach in such a collaborative environment consist in finding a regular expression exp from DB such that $matching(exp, R) = TRUE$ while ensuring that none of the databases DB_i provide any information from its content to anyone.

1.3 The SREXM approach

This section will provide an overview of the secure architecture SREXM (*Secure Regular Expression Mapping*). It is to answer the problem of privacy preserving in a collaborative environment. Inspired by the work of [9], this architecture offers the advantage of achieving the various operations while ensuring that neither party may have access to private data contained in the initial databases. In addition to the client site S which is responsible to provide the request to be tested, the architecture

I	$cond_1$	S_1	1	$cond_1$	/
S_1	$cond_2$	I	2	$cond_2$	..
S_1	$cond_3$	S_2	1	$cond_3$	[^./e]
S_1	$cond_4$	S_3	1	$cond_4$	e
S_2	$cond_5$	S_2	1	$cond_5$	[^./]
S_2	$cond_6$	I	3	$cond_6$	/..
S_3	$cond_7$	S_2	1	$cond_7$	[^./t]
S_3	$cond_8$	S_4	1	$cond_8$	t
S_4	$cond_9$	S_2	1	$cond_9$	[^./c]
S_4	$cond_{10}$	S_5	1	$cond_{10}$	t
S_5	$cond_{11}$	S_2	1	$cond_{11}$	[^./]
S_5	$cond_{12}$	S_6	1	$cond_{12}$	/
S_6	$cond_{13}$	I	2	$cond_{13}$	..
S_6	$cond_{14}$	F	6	$cond_{14}$	passwd

Fig. 1.1 Automaton associated to the Regular Expression *exp*

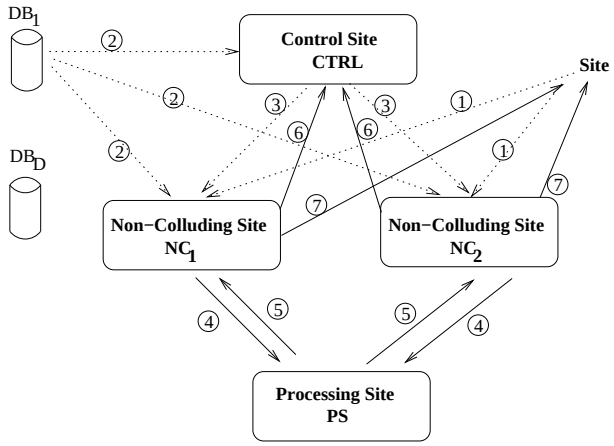

Fig. 1.2 General Architecture of SREXM

requires four non-collaborative and semi honest sites [4]: they follow the protocol correctly, but are free to use the information they have collected during the execution of the protocol. These independent sites collect, store and evaluate information in a secure way. The different functions provided by these sites are:

- **The Control Site** $CTRL$: $CTRL$ is used to rule the various operations needed to match the regular expression. To do this, it interacts with the two non colluding sites NC_1 and NC_2.
- **Non Colluding Sites** NC_1 **and** NC_2: These two symmetric sites collect garbled data from all databases as well as the garbled request to be tested from S. Under the control of $CTRL$ and by interaction with PS, they perform several secure

operations in order to insure that none of them will be able to infer any of the intermediate results or the final result which is returned to site S.

- **The Processing Site** PS: This site is used both by NC_1 and NC_2 to process, the various operations needed, in a secure way. Like NC_1 and NC_2, PS also cannot deduce any pertinent value of intermediate or final result from the data it processes.

The exchange of data between the different sites is done by using the secure method $SEND^S(\overset{+}{v}|\overset{-}{v})'$ which sends the vector of bits V = $\overset{+}{v} \oplus \overset{-}{v}$ to NC_1 and NC_2. It is defined in order to send $\overset{+}{v}$ to NC_1 and $\overset{-}{v}$ to NC_2 (or vice versa). A random vector R is used for secure transmission such that $\overset{+}{v} = R$ and $\overset{-}{v} = V \oplus R$. This method is used in particular to send the data from the databases DB_i and to send the request from site S. Thus, the process described in figure 1.2 starts in the following way. First, the site S sends its request to NC_1 and NC_2 using the $SEND^S$ method (See arrow number 1 in figure 1.2). More precisely, the request R is taken in its boolean form: a vector of bits. A random vector of bits A_R is then generated with the same size as the request R to compute the new vector $Z_R = A_R \bigoplus R$. Z_R is sent to NC_1 and A_R to NC_2 (or vice versa). Each database DB_i decompose the transition matrix in three tables: the first contains the transitions of the automaton, the second the conditions and the third the lengths of the shifts. To encode the transition matrix, the indexes of these tables are randomly mixed. The databases first send the table of transition to $CTRL$, then, using $SEND^S$, the associated tables of conditions and lengths are sent to NC_1 and NC_2 in the same order of indexes as the one used when sent to $CTRL$ (See arrow number 2). From this point, the computation of the request is done under the control of $CTRL$. Via the $NCOMPARE^S$, it will ask NC_1 and NC_2 to test the condition of index i from the table of conditions (See arrow number 3). At this point, NC_1 has part of the request to be tested $\overset{+}{R}$, part of the condition $\overset{+}{ST}_{R_i}$ and the current position pos in the request R. In the same way, NC_2 has $\overset{-}{R}$, $\overset{-}{ST}_{R_i}$ and the position pos in the request. Then NC_1 et NC_2 just have to extract the substring of the request starting at position pos and of same length the string to compare ($\overset{+}{ST}_{R_i}$ or $\overset{-}{ST}_{R_i}$). The next step consist in the comparison of the two string in a secure way. This is performed by sending requested data to PS using the $NCMP^S$ protocol (See arrow number 4). Under completion, the result of the comparison is divided in two parts, one is owned by NC_1 and the other by NC_2 such that none are able to infer its real value. Both parts are then securely returned to $CTRL$ (see arrows 5 and 6) which uses the result to change the state of the automaton. The process is repeated under control of $CTRL$ unless the automaton is ended (it moves to a final state of the automaton or the request does not match). The action of maintaining the position pos in the request is done by $CTRL$ through the secure operation $INCR^S$ whose aim is to shift the position according to the displacement length associated with the transition. This is done by sending the index in the table of lengthes to NC_1 and NC_2 that will update the value of pos. When the automaton reaches a final state, $CTRL$ or matching of the request fails, $CTRL$ aggregates the results (attack, non-attack, unknown) using the secure method $AGGREGATE^S$. The aggregated result

is split between NC_1 and NC_2 and kept while data has to be performed on the same request. At the end of the process, the final aggregated result is sent to S.

1.4 The Secure Algorithms

In this section, we present the various algorithms used in SREXM approach. In order to simplify writing, we consider the following notations: Let $(\overset{+}{X}|\overset{-}{X}) \leftarrow h^S(\overset{+}{Y_1}\ldots\overset{+}{Y_n}|\overset{-}{Y_1}\ldots\overset{-}{Y_n})$ be a tripartite computation of any function h^S between NC_1, NC_2 and PS where NC_1 owned some of the entries $\overset{+}{Y_1}\ldots\overset{+}{Y_n}$ and gets part of the result $\overset{+}{X}$ and similarly NC_2 owned some of the entries $\overset{-}{Y_1}\ldots\overset{-}{Y_n}$ and gets part of the result $\overset{-}{X}$. The final result is obtained by applying the binary operator XOR ($\bigoplus$) between $\overset{+}{X}$ and $\overset{-}{X}$. However, this does not mean that NC_1 sends exactly $\overset{+}{Y_1}\ldots\overset{+}{Y_n}$ to PS an receives the result $\overset{+}{X}$ from PS. In fact, NC_1 garbles its inputs $\overset{+}{Y_1}\ldots\overset{+}{Y_n}$ by adding random noise and gets $\overset{+}{Y'_1}\ldots\overset{+}{Y'_n}$ which are securely sent to PS. Similarly, NC_2 sends its garbled inputs to PS. At the end of the process, both sites receive a part of garbled result from PS (respectively $\overset{+}{X'}$ and $\overset{-}{X'}$). This intermediate result may now be used as input of further computation. We will also use the following simplifications:

1. $g^S(\overset{+}{x},\overset{+}{y}|\overset{-}{x},\overset{-}{y}) \Leftrightarrow g^S(\overset{+}{x}|\overset{-}{x};\overset{+}{y}|\overset{-}{y})$
2. Si $h^S()$ is a 2 argument function then $h^S(\overset{+}{X_1},\cdots,\overset{+}{X_n}|\overset{-}{X_1},\cdots,\overset{-}{X_n})$ will correspond to $h^S(h^S(\cdots h^S(h^S(\overset{+}{X_1},\overset{+}{X_2}|\overset{-}{X_1},\overset{-}{X_2});\overset{+}{X_3}|\overset{-}{X_3})\cdots);\overset{+}{X_n}|\overset{-}{X_n})$

1.4.1 The Algorithm $NCOMPARE^S$

Algorithm 1: Algorithm $NCOMPARE^S$

Data: $(i|i)$ The index of the condition to be tested is sent to NC_1 and NC_2 by $CTRL$.

Result: $(\overset{+}{b}|\overset{-}{b})$ two booleans such that $b = \overset{+}{b} \oplus \overset{-}{b}$ is false when STR_i matches the substring starting a current position of the request. Otherwise, it is true.

1. NC_1 ccomputes $Len_1 = length(\overset{+}{STR_i})$; NC_2 ccomputes $Len2 = length(\overset{-}{STR_i})$.
   // By definition $Len_1 = Len_2$
2. If ($(pos + Len_1 > length(\overset{+}{R}))$——$(pos + Len_2 > length(\overset{-}{R}))$)
   then return $(\overset{+}{b}|\overset{-}{b}) = (1|0)$
3. NC_1 computes $\overset{+}{S} = \overset{+}{R}_{pos}\cdots\overset{+}{R}_{pos+Len_1-1}$
4. NC_2 computes $\overset{-}{S} = \overset{-}{R}_{pos}\cdots\overset{-}{R}_{pos+Len_2-1}$
5. compute $(\overset{+}{b}|\overset{-}{b}) = NCMP^S(\overset{+}{S}|\overset{-}{S})$ using PS, NC_1 and NC_2.

The evaluation of the condition $NCOMPARE^S(Str|Str)$ (See Algorithm 1) associated with a transition is controlled by the controller $CTRL$. It sends the index i of a string in the table of conditions to NC_1 and NC_2. Thus NC_1 only hold the part $\overset{+}{STR}_i$ and NC_2 the other part $\overset{-}{STR}_i$ such that the real string is $STR_i = \overset{+}{STR}_i \oplus \overset{-}{STR}_1$. Each NC_1 and NC_2 sites also holds its part of the request $(\overset{+}{R}|\overset{-}{R})$ and the current position in the request (*pos*). After extracting the substring of the request R starting at position *pos* and of same length with STR_i, the comparison is performed via $NCMP^S$. The operator $NCMP^S(\overset{+}{S_1},\overset{+}{S_2}|\overset{-}{S_1},\overset{-}{S_2}) \rightarrow (\overset{+}{b}|\overset{-}{b})$ (see section 1.4.4) compares two sequence of bits of same length $S_1 = \overset{+}{S_1} \oplus \overset{-}{S_1}$ and $S_2 = \overset{+}{S_2} \oplus \overset{-}{S_2}$ and returns a boolean value $b = \overset{+}{b} \oplus \overset{-}{b}$ such that b is false if S_1 and S_2 are identical and otherwise true. The final result is returned to $CTRL$.

Complexity: The complexity of $NCOMPARE^S$ is same as the one of $NCMP^S$ (see section 1.4.4).

$COMPARE^S$ does not allow NC_1 or NC_2 to get knowledge on the result of the comparison. They can only deduce the length of left part of the request which have been successfully matched by the automaton (in fact the value of *pos*). But even if they could obtain the list of strings that has been matched successfully, as they only hold random data in the table of condition, they can only infer that a random sequence of length *pos* has matched the beginning of the request. However, they can not deduce neither whether the filtering was successful or not nor the value associated with the final state in case of successful filtering. At the level of $CTRL$ no information on the length of the filtered part of the query can be inferred. Indeed $CTRL$ has no access to the real data (request, condition strings, lengthes). It only knows indexes. The only information it can obtain is the path followed by the automaton to provide an answer.

1.4.2 The algorithm $INCR^S$

The request R to be tested is split between NC_1 and NC_2, in a secure way. The starting position *pos* is known by both NC_1 and NC_2. Any modification to this position is controlled by $CTRL$ via the $INCR^S(len|len)$ operation. When the automaton is sent to $CTRL$ and data to NC_1 and NC_2, these two also receives a table with indexes aleatory sorted and which contains the lengths of movements. The goal of this sort is to avoid any direct correspondence between the index of conditions and lengths. The $INCR^S$ method just sends the index to be used to NC_1 and NC_2 and each one updates the position *pos* according to the value found in the table.

When an increment is triggered by $CTRL$, there is no way for NC_1 or NC_2 to know which condition had activated it. In fact $CTRL$ may execute unnecessary computation. On $CTRL$ side, neither the information of the length may be available nor inferred as it knows only indexes.

1.4.3 The algorithm $AGGREGATE^S$

Aggregation of results simply consists to securely retain the first valid result obtained by *CTRL*, *i.e.* when an automaton has matched the request and lead in a final state. The objective of $AGGREGATE^S$ is to conceal from NC_1 and NC_2, the fact that an automaton has filtered the request (and associated value W_f) or not. This is done by setting a state bit to 1 if the automaton has filtered the request and 0 otherwise. Depending on the value of this bit, the information stored in the accumulator between NC_1 and NC_2 will be either the value of the final state W_f or a random vector. The implementation of $AGGREGATE^S$ require the secure operators $\bigvee^S(\overset{+}{s_1},\overset{+}{s_2}|\overset{-}{s_1},\overset{-}{s_2}) \rightarrow (\overset{+}{v}|\overset{-}{v})$ and $\bigwedge^S(\overset{+}{s_1},\overset{+}{s_2}|\overset{-}{s_1},\overset{-}{s_2}) \rightarrow (\overset{+}{v}|\overset{-}{v})$ which implements respectively a secure computation of bitwise operators OR and AND on vectors of bits of same length (S_1 and S_2) and returns the sequence V. At the end of the process SREXM, NC_1 and NC_2 both sends the value of their part of the accumulator to the client site *S*. Finally *S* has just need to take XOR of the received values to get the result.

For each regular expression (automaton), the values V_f associated with final states are encoded with random numbers R_1 and R_2 by computing $W_f = V_f \oplus R_1 \oplus R_2$. *CTRL* knows W_f, NC_1 knows R_1 and NC_2 knows R_2. We consider that the length of W_F is identical in all databases.

Algorithm 2: Algorithm $AGGREGATE^S$

Data: $Y = \overset{+}{Y} \oplus \overset{-}{Y}$ of length $n+1$ whose first bit is the bit of state set by *CTRL*.
// $\overset{+}{A}$ and $\overset{-}{A}$ are the aggregated values respectively kept by NC_1 and NC_2.
// $n+1$ is the length of A.

1. NC_1 computes $\overset{+}{Z} = \overset{+}{Y} \oplus 0R_1$; NC_2computes $\overset{-}{Z} = \overset{-}{Y} \oplus 0R_2$;
2. $\forall k \in 1..n$ NC_1,NC_2 and *PS* compute
   $(\overset{+}{B_k}|\overset{-}{B_k}) = \bigwedge^S(\ \bigvee^S(\overset{+}{Z_0},\overset{+}{A_k}|\overset{-}{Z_0},\overset{-}{A_k})\ ;\ \bigvee^S(\neg\overset{+}{Z_0},\overset{+}{Z_k}|\overset{-}{Z_0},\overset{-}{Z_k})\ )$
3. NC_1, NC_2 and *PS* compute $(\overset{+}{B_0}|\overset{-}{B_0}) = \bigvee^S(\overset{+}{Z_0},\overset{+}{A_0}|\overset{-}{Z_0},\overset{-}{A_0})$
4. NC_1 and NC_2 respectively computes $\overset{+}{A} = \overset{+}{B}$ and $\overset{-}{A} = \overset{-}{B}$.

Property 1 *$AGGREGATE^S$ prohibits NC_1 and NC_2 to access the value stored in the accumulator. They even do not know if the value stored in the accumulator has changed or not.*

Proof: The data $(\overset{+}{Y}|\overset{-}{Y})$ held by NC_1 and NC_2 are randomized by *CTRL*. It is therefore impossible to know the value of Y and obviously that of Y_0 (*i.e.* the state bit indication whether the automaton has reached a final state or not). As operators $\bigvee^S$ and $\bigwedge^S$ returns values garbled with random noise, from the point of view of NC_1 (respectively NC_2) the received value $\overset{+}{B}$ (respectively $\overset{-}{B}$) is pure random and thus independent from the values of $\overset{+}{Y}$ and $\overset{+}{A}$ (respectively $\overset{-}{Y}$ and $\overset{-}{A}$). In particular, although NC_1 and NC_2 know the initial value of A (0 at the beginning of the process), it is impossible for them to deduce whether this value has been changed or not, once

$AGGREGATE^S$ has been used.

Complexity: The methods $\bigvee^S$ and $\bigwedge^S$ are used $2n+1$ and n times respectively on one bit. By reusing the complexity of operators $\bigvee^S$ and $\bigwedge^S$ (see section 1.4.4), NC_1 and NC_2 therefore perform $34n+12$ binary operations, generate $6n+2$ aleatory bits, send $12n+4$ bits and receive $10n+4$ bits (including parameters). *PS* performs $12n+4$ binary operations, generates $3n+1$ aleatory bits, receives $12n+4$ bits ($6n+2$ from NC_1 and NC_2 each) and sends $6n+2$ bits ($3n+1$ to NC_1 and NC_2 each). Obviously this has to be compared with the length of inputs ($n+1$ bits).
Remarks: The two mechanisms *bufferization of data sent by the databases* and *aggregation of results* fulfil databases anonymization. Indeed, even if the client can identify which databases are sending data to SREXM, it can not infer the one which gave the final result. The aggregated value may be returned to the client immediately after a valid match. However, in this case, NC_1 and NC_2 are able to infer the identity of the database who gave the answer. To improve the anonymization, it is necessary to wait, for example until each data from all databases have been processed. Meanwhile this approach is secure, but it is unfortunately not effective because too expensive in term of time. To minimize time cost, we can return intermediate values to the clients each time n results are aggregated which lower the time overcost to $n/2$. In fact both anonymization mechanisms have different costs: the buffering essentially introduces space cost while aggregation introduces computing time cost. It is of course possible to mix the two mechanism and adapt parameters to adjust anonymization process according to the needs and bearable costs.

Algorithm 3: The Algorithm $\bigwedge^S$

Data: $(\overset{+}{X},\overset{+}{Y}|\overset{-}{X},\overset{-}{Y})$ vector of bit/s are such that $\overset{+}{X}$ and $\overset{+}{Y}$ are in NC_1, and $\overset{-}{X}$ and $\overset{-}{Y}$ are in NC_2

Result: $(A^R|B^R)$ is such that $A^R \oplus B^R = (\overset{+}{X} \oplus \overset{-}{X}) \bigwedge (\overset{+}{Y} \oplus \overset{-}{Y})$

1. NC_1 and NC_2 mutually generate and exchange four random vectors of bits R_A, R'_A, R_B and R'_B such that: $\overset{+}{X}' = \overset{+}{X} \bigoplus R_A$, $\overset{+}{Y}' = \overset{+}{Y} \bigoplus R'_A$, $\overset{-}{X}' = \overset{-}{X} \bigoplus R_B$ and $\overset{-}{Y}' = \overset{-}{Y} \bigoplus R'_B$.
2. NC_1 sends $\overset{+}{X'}$ and $\overset{+}{Y'}$ to *PS*.
3. NC_2 sends $\overset{-}{X'}$ and $\overset{-}{Y'}$ to *PS*.
4. *PS* computes $\overset{+}{C} = \overset{+}{X'} \bigwedge \overset{-}{Y'}$ and $\overset{-}{C} = \overset{+}{Y'} \bigwedge \overset{-}{X'}$ and generates a random vector of bit/s R_{PS}.
5. *PS* sends $A'_{PS} = \overset{+}{C} \bigoplus R_{PS}$ to NC_1 and $B'_{PS} = \overset{-}{C} \bigoplus R_{PS}$ to NC_2.
6. NC_1 computes $A^R = A'_{PS} \bigoplus (\overset{+}{X} \bigwedge R'_B) \bigoplus (\overset{+}{Y} \bigwedge R_B) \bigoplus (\overset{+}{X} \bigwedge \overset{+}{Y}) \bigoplus (R_B \bigwedge R'_A)$
7. NC_2 computes $B^R = B'_{PS} \bigoplus (\overset{-}{X} \bigwedge R'_A) \bigoplus (\overset{-}{Y} \bigwedge R_A) \bigoplus (\overset{-}{X} \bigwedge \overset{-}{Y}) \bigoplus (R_A \bigwedge R'_B)$.

1.4.4 The algorithms $NCMP^S$, $\bigwedge^S$ and $\bigvee^S$

In this section, we define three algorithms used to implement the secure operator for string comparison, the basic principle of these algorithms is to add uniform random noise to the data which could be deleted from the final result.

The $\bigwedge^S$ protocol begins with NC_1 and NC_2 who modify their data by doing XOR them with random values (see step 1 in algorithm). NC_1 and NC_2 share these random values (also see step 1). Garbled data are then send to PS (step 2 and 3) which is now able to compute $\bigwedge$ in a secure way (step 4). In fact, PS gets only garbled inputs indistinguishable from random and unrelated to each others and thus calculates random values from its point of view. To avoid NC_1 and NC_2 from inferring the final result, it does XOR with random noise to the values it calculates before sending them back to NC_1 and NC_2 (step 5). Now NC_1 and NC_2 may both obtain their part of the final result by removing the random noise they added on step 1 (see step 6 and 7). The final result is obtained bay computing $A^R \bigoplus B^R = A'_{PS} \bigoplus (\overset{+}{x} \bigwedge R'_B) \bigoplus (\overset{+}{y} \bigwedge R_B) \bigoplus (\overset{+}{x} \bigwedge \overset{+}{y}) \bigoplus (R_B \bigwedge R'_A) \bigoplus B'_{PS} \bigoplus (\bar{x} \bigwedge R'_A) \bigoplus (\bar{y} \bigwedge R_A) \bigoplus (\bar{x} \bigwedge \bar{y}) \bigoplus (R_A \bigwedge R'_B)$ o $A'_{PS} \bigoplus B'_{PS} = (\overset{+}{x} \bigwedge R'_B) \bigoplus (\overset{+}{y} \bigwedge R_B) \bigoplus (\bar{x} \bigwedge R'_A) \bigoplus (\bar{y} \bigwedge R_A) \bigoplus (\overset{+}{x} \bigwedge \overset{+}{y}) \bigoplus (\bar{x} \bigwedge \bar{y}) \bigoplus (R_A \bigwedge R'_B) \bigoplus (R_B \bigwedge R'_A) \bigoplus R_{PS} \bigoplus R_{PS}$.

Using the property of the XOR operator: $R \bigoplus R = 0$, we get the desired result: $A^R \bigoplus B^R = \overset{+}{x} \bigwedge \overset{+}{y} \bigoplus \overset{+}{x} \bigwedge \bar{y}\, \bar{x} \bigwedge \overset{+}{y} \bigoplus \bar{x} \bigwedge \bar{y}$. Which is a re-written form of $(\overset{+}{x} \bigoplus \bar{x}) \bigwedge (\overset{+}{y} \bigoplus \bar{y})$. However, this operation is never performed by the non collaborative sites and the final result is kept shared between NC_1 and NC_2.

The $\bigvee^S$ protocol is identical to the $\bigwedge^S$ protocol except for the last two steps (steps 6 and 7) performed by NC_1 and NC_2. Thus we get the final result: $A^R \bigoplus B^R = \overset{+}{c'} \bigoplus (\overset{+}{x} \bigwedge R'_B) \bigoplus (\overset{+}{y} \bigwedge R_B) \bigoplus \overset{+}{x} \bigoplus \overset{+}{y} \bigoplus (\overset{+}{x} \bigwedge \overset{+}{y}) \bigoplus (R_B \bigwedge R'_A) \bigoplus \bar{c'} \bigoplus (\bar{x} \bigwedge R'_A) \bigoplus (\bar{y} \bigwedge R_A) \bigoplus \bar{x} \bigoplus \bar{y} \bigoplus (\bar{x} \bigwedge \bar{y}) \bigoplus (R_A \bigwedge R'_B)$. This reduce to the desired result: $A^R \bigoplus B^R = \overset{+}{x} \bigoplus \overset{+}{y} \bigoplus (\overset{+}{x} \bigwedge \overset{+}{y}) \bigoplus (\overset{+}{x} \bigwedge \bar{y}) \bigoplus \bar{x} \bigoplus \bar{y} \bigoplus (\bar{x} \bigwedge \overset{+}{y}) \bigoplus (\bar{x} \bigwedge \bar{y})$.

Which is a re-written form of $(\overset{+}{x} \oplus \bar{x}) \bigvee (\overset{+}{y} \oplus \bar{y})$.

Algorithm 4: The algorithm $\bigvee^S$

Data: $(\overset{+}{x}, \overset{+}{y} | \bar{x}, \bar{y})$ vectors of bits such that $\overset{+}{x}$ et $\overset{+}{y}$ belongs to NC_1, $\bar{x}$ and $\bar{y}$ belongs to NC_2.

Result: $(A^R | B^R)$ is such that $A^R \bigoplus B^R = (\overset{+}{x} \bigoplus \bar{x}) \bigvee (\overset{+}{y} \bigoplus \bar{y})$.

1..5. These steps are same as initial 5 steps of $\bigwedge^S$ function.

6. NC_1 computes $A^R = A'_{PS} \bigoplus (\overset{+}{x} \bigwedge R'_B) \bigoplus (\overset{+}{y} \bigwedge R_B) \bigoplus \overset{+}{x} \bigoplus \overset{+}{y} \bigoplus (\overset{+}{x} \bigwedge \overset{+}{y}) \bigoplus (R_B \bigwedge R'_A)$.
7. NC_2 computes $B^R = B'_{PS} \bigoplus (\bar{x} \bigwedge R'_A) \bigoplus (\bar{y} \bigwedge R_A) \bigoplus \bar{x} \bigoplus \bar{y} \bigoplus (\bar{x} \bigwedge \bar{y}) \bigoplus (R_A \bigwedge R'_B)$.

Property 2 *$\bigwedge^S$ and $\bigvee^S$ forbid NC_1 to gain any information of private data of NC_2 (and vice versa). Moreover, the PS learns none of their private inputs.*

Proof: From the protocol, B'_{PS} is the only value that NC_2 can learn from the private data of NC_1. Due to the noise, R_{PS}, added by PS, NC_2 is still not able to deduce

the values of $\overset{+}{X}$ or $\overset{+}{Y}$. As the roles of NC_1 and NC_2 are interchangeable, the same argument holds for NC_1, not able to learn the private inputs $\bar{X}$ or $\bar{Y}$ of NC_2. However, one key security aspect of not leaking any information to *PS* is achieved by randomizing the inputs before transmitting them to the Processing Site. Due to the randomization performed during the initial step, it just infers a stream of uniformly distributed values, and cannot distinguish between a genuine and a random value.

Complexity: Length of bit vector is 1: For the operator $\bigwedge^S$, NC_1 and NC_2 each performs 10 binary operations ($6\bigoplus$ and $4\wedge$). $\bigvee^S$ does two more $\bigoplus$ that means 12 binary operations. For both operators NC_1 and NC_2 generate 2 random bits, exchange 2×2 random bits and send 2×1 bits to *PS*. *PS* generates 1 random bit and performs 4 binary operation ($2\bigoplus$ and $2\bigwedge$) and returns 2 bits to NC_1 and NC_2 each.

The $NCMP^S()$ method compares two vectors of bits by using the secure $\bigvee^S$ method. The result of $NCMP^S()$ consists of 2 bits. One is sent to NC_1 and the other is sent to NC_2. XOR of these two bits is 0 if the vectors are similar, otherwise 1.

Algorithm 5: The Algorithm $NCMP^S$

Data: Half part of V and W is owned by NC_1 and the other part is owned by NC_2

Result: $(\overset{+}{R}|\bar{R})$ is such that $\overset{+}{R}\oplus\bar{R}=0$ if $V=W$ else 1

1. NC_1 computes $X\leftarrow\overset{+}{V}\oplus\overset{+}{W}$ where $X=(X_1,X_2,\cdots,X_l)$ and l is the length of vector V and W.
2. NC_2 computes $Y\leftarrow\bar{V}\oplus\bar{W}$ where $Y=(Y_1,Y_2,\cdots,Y_l)$.
3. $(\overset{+}{R}|\bar{R})\leftarrow OR^S(X_1,X_2,\cdots,X_l|Y_1,Y_2,\cdots,Y_l)$

Complexity: Length of bit vector is l: CMP^S executes l $\oplus$ operations and $l-1$ $\bigvee^S$. Thus NC_1 and NC_2 compute $13l-12$ binary operations, generate $2l-2$ aleatory bits, receive $4l-3$ bits (including inputs) and send $5l-4$ bits (including the result). On *PS* side, *PS* computes $4l-4$ binary operations, generates $l-1$ aleatory bits, receives $4l-4$ bits and sends $2l-2$ bits.

Property 3 *NC_1 and NC_2 gain no information of the real values which are compared and of the result of the comparison.*

Proof: The input data sent to NC_1 and NC_2 are garbled with random values. Thus they cannot distinguish them from random values. In the same way, all values returned by $\bigvee^S$ are also garbled with unrelated random bits. Thus NC_1 and NC_2 only gets random values and then cannot infer the actual values of the inputs or results. If *PS* keeps history of intermediate results, it might deduce a part of the aleatory bits that were used to encode its results sent to NC_1 and $NC2$. However, this gives no information of actual data.

1.5 Conclusion

In this paper, we proposed a new approach of secured intrusion detection in a collaborative environment. Via our approach an application can use knowledge from foreign databases to identify whether a request corresponds to an attack or not. We have demonstrated that the proposed architecture ensured that it is impossible to identify which database has given the answer and that none of the internal components of the architecture can infer knowledge on the databases or on the request from the data they got. Our approach may also provide the type of the attack when they are specified in the databases. Our current work concern the study of the removal of the fourth semi-honest site $CTRL$ by trying to dispatch its proceedings on the automaton on the three other ones. In parallel, we try to improve the management of the automaton (*i.e.* introduce more powerful comparison operators).

References

1. F. Cuppens and A. Miege. Alert correlation in a cooperative intrusion detection framework. In *Proc. of the IEEE International Conference on Networks (ICON 2005)*, pages 118–123, 2005.
2. The Open Source Vulnerability Database. http://osvdb.org/. 2008.
3. T. Escamilla. *Intrusion Detection: Network Security beyond the firewall*. John Wiley and Sons, New York, 1998.
4. O. Goldreich. Secure multi-party computation - working draft. *citeseer.ist.psu.edu/goldreich98secure.html*, 2000.
5. R. Graham. FAQ: Network intryusion detection system. *http://www.robertgraham.com/pubs/network-intrusion-detection.html*, 2001.
6. R. Heady, G. Luger, A. Maccabe, and M. Servilla. The architecture of a network level intrusion detection system. *Technical Report CS9020*, 1990.
7. J.E. Hopcroft, R. Motwanu, Rotwani, and J.D. Ullman. *Introduction to Automata Theory, Languages and Computability*. Addison-Wesley, 2000.
8. R. Janakiraman, M. Waldvoge, and Q. Zhang. Indra: a peer-to-peer approach to network intrusion detection and prevention. In *Proc. of the 12th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 226–231, 2003.
9. M. Kantarcioglu and J. Vaidya. An architecture for privacy-preserving mining of client information. In *Proc. of the Workshop on Privac*, pages 27–42, 2002.
10. M. Locasto, J. Parekh, A. Keromytis, and S. Stolfo. Towards collaborative security and p2p intrusion detection. In *Proceedings of the 2005 IEEE Workshop on Information Assurance and Security*, West Point, NY, 2005.
11. J. McHugh, A. Christie, and J. Allen. Defending yourself: the role of intrusion detection systems. *IEEE Software*, pages 42–51, Sep/Oct 2000.
12. P.E. Proctor. *Practical Intrusion Detection Handbook*. Prentice-Hall, 2001.
13. K. Wang, G. Cretu, and S. Stolfo. Anomalous payload-based worm detection and signature generation. In *Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection*, 2005.
14. Guangsen Zhang and Manish Parashar. Cooperative defence against ddos attacks. *Journal of Research and Practice in Information Technology*, 38(1), 2006.
15. C. V. Zhou, S. Karunasekera, and C. Leckie. Evaluation of a decentralized architecture for large scale collaborative intrusion detection. In *Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Network Management (IM 2007)*, pages 80–89, 2007.