Anti-correlation as a criterion to select appropriate counter-measures in an intrusion detection network

Frédéric Cuppens, Fabien Autrel, Yacine Bouzida, Joaquin Garcia-alfaro, Sylvain Gombault, Thierry Sans

Anti-correlation as a criterion to select appropriate counter-measures in an intrusion detection network

Frédéric CUPPENS*, Fabien AUTREL*, Yacine BOUZIDA*,
Joaquin GARCIA*,**, Sylvain GOMBAULT*, Thierry SANS*

Abstract

Since current computer infrastructures are increasingly vulnerable to malicious activities, intrusion detection is necessary but unfortunately not sufficient. We need to design effective response techniques to circumvent intrusions when they are detected. Our approach is based on a library that implements different types of counter-measures. The idea is to design a decision support tool to help the administrator to choose, in this library, the appropriate counter-measure when a given intrusion occurs. For this purpose, we formally define the notion of anti-correlation which is used to determine the counter-measures that are effective to stop the intrusion. Finally, we present a platform of intrusion detection that implements the response mechanisms presented in this paper.

Key words: Computer security, Intruder detector, Information protection, Correlation, Modeling, Logic model.

L'ANTI-CORRÉLATION COMME CRITÈRE DE SÉLECTION DE CONTRE-MESURES DANS LE CONTEXTE DE LA DÉTECTION D'INTRUSION

Résumé

Étant donné que les systèmes informatiques sont de plus en plus vulnérables aux activités malveillantes, l'utilisation de la détection d'intrusion est nécessaire mais ne suffit pas. Nous devons élaborer des méthodes efficaces de réaction aux intrusions afin d'arrêter les intrusions détectées. Notre approche est basée sur une bibliothèque de différents types de contre-mesures. L'objectif est d'aider l'administrateur à choisir dans cette bibliothèque la contre-mesure la mieux adaptée quand une intrusion est détectée. Pour ce faire nous définissons formellement la notion d'anti-corrélation qui est utilisée pour sélectionner les contre-mesures permettant d'arrêter l'intrusion. Nous finissons par la présentation d'une plateforme de détection d'intrusion mettant en œuvre les mécanismes présentés dans cet article.

Mots clés : Sécurité informatique, Détecteur intrus, Protection information, Corrélation, Modélisation, Modèle logique.

* GET-ENST Bretagne, 2, rue de la Châtaigneraie, CS 17607, 35576, Cesson Sévigné Cedex, France
** UAB-DEIC, Edifici Q, 08193 Bellaterra, Spain
{fcuppens, fautrel, ybouzida,sgombault,tsans}@rennes.enst-bretagne.fr, jgarcia@deic.uab.es

Contents

I. INTRODUCTION

Current systems that compose distributed computer infrastructures are increasingly vulnerable to intrusions and malicious activities. Several approaches have been suggested to detect such intrusions [1, 12, 14]. However, it is generally considered that current intrusion detection systems produce very large volume of alerts, including true alerts but also many false positives (see [4, 10] for instance). This is why recent research work attempts to understand and model intrusion strategies to provide a more global and precise diagnostic of the intrusion [6, 11]. These approaches are interesting and represent a step in the right direction but detecting intrusion is not sufficient. It is also necessary to develop automated defenses capable of appropriate responses to counter intrusions when they occur.

Several response strategies are possible including launching counter measures against the intruder to prevent his or her malicious activity to proceed or act on the target system to stop the intrusion and recover in a safe state. Direct responses against the intruder is a complex problem that includes several technical difficulties (in particular, it is necessary to precisely identify the origin of the intrusion) and legal and ethic complications (directly acting on the intruder is generally viewed as illegal activities). In this paper, we shall not consider this type of response and actually focus on responses that consist in acting on the target system.

When an intrusion occurs, the appropriate response on the target system generally depends on the type of intrusion being performed. For instance, the response will not be the same in the case of a denial of service (DOS) attack or a user to root (U2R) attack. Thus, our approach is based on a library of responses that contains different types of possible counter-measures which may be launched to stop intrusions. The problem addressed in this paper is to choose the appropriate counter-measure when a given intrusion occurs. This may represent a complex task for the administrator to make such a choice and some support might be useful to help the administrator. It is also necessary to fix the parameters of the response. For instance when the response consists in closing a given connection, the IP addresses of the source and target must be appropriately fixed before launching the response.

In this paper, we suggest an approach to define decision mechanisms to help the administrator to choose, in the response library, the counter-measure candidates when an intrusion occurs and to present them to the administrator with the appropriate parameters to circumvent the intrusion. Once the administrator selects a counter-measure, this counter-measure along with the appropriate parameters is automatically executed to stop the intrusion in the target system.

Our approach is based on a logical formalization of both attacks and counter-measures. This formalism is used to derive, from the attack description (especially the effects of an attack on the target system), one or several counter-measures that may circumvent the attack.

For this purpose, we define the notion of *anti-correlation.* This notion is used to determine the counter-measures that will have a negative effect on the attack and therefore will enable the administrator to stop this attack.

The remainder of this paper is organized as follows. Section II presents the notion of response and suggests several types of response. In Section III, we present our formalism to model attacks and counter-measures. Our formalism is based on LAMBDA, a language suggested in [7] to model attacks. In this section, we also suggest using LAMBDA to model counter-measures. Section IV recalls the definition of correlation [6] and introduces the notion of anti-correlation. In Section V, we show how to use anti-correlation to determine relevant counter-measures (1) to act on the objective of an intrusion or (2) to cut an ongoing attack scenario by acting on a given step of this scenario. We also present how our approach provides means to parameterize the selected counter-measures. Section VI gives an example to illustrate the approach and presents an intrusion detection platform that includes the response mechanisms suggested in this paper. Finally, Section VII concludes the paper and suggests several possible extensions to our approach.

This paper is an extended version of the previous work presented at SAR 04 [5]. In particular, both the intrusion detection framework and the examples given in Section VI have been revised and enhanced.

II. RESPONSE MECHANISM AND COUNTER-MEASURES

Even though some improvements were made recently, current Intrusion Detection Systems (IDS) propose few response mechanisms in addition to alerts and reports. There is only a small variety of response techniques and the decision criteria that are used to activate the response remain often simplistic. Moreover, in a context of exploitation, security administrators generally balk at using the most interesting responses like automatic reconfiguration of firewalls or routers. This is due to lack of confidence in the capabilities of the IDS to take the right decision. Administrators also fear of not controlling the consequences of the automation of counter-measures.

Lastly, the objective of most responses consists in stopping an ongoing attack. More elaborate responses that are effective to automatically correct the detected vulnerabilities, remain marginal.

In [9], the following taxonomy of counter-measures was suggested:

– Information: Specific action that raises an alert for the security administrator. This action can be launched after detecting an intrusion of sufficient severity.
– Deterrence: Action performed against the intruder so that he or she will be willing to stop his or her malicious activity. For instance, a message sent to the intruder to notify that his or her malicious action were detected is the simplest (but not always effective) form of deterrence.
– Correction: Action to modify the system state to correct an identified vulnerability or a system misconfiguration with respect to the security policy. For instance, installing a patch is a form of correction.
– Compensation: Action performed to block the attack but without correction. The system is still vulnerable but the response prevents the intruder from performing his or her

intrusion. For instance, it is possible to stop a vulnerable service, or close the connection between the intruder and the target (using a TCP-Reset), or reconfigure a firewall to block the attack source.

We accept this taxonomy but we make a difference between actions that change the state of the system to be protected and other actions that do not cause such a change. In the remainder of this paper, we shall actually consider actions that change the system state, that is correction and compensation. Since information and deterrence have respectively an effect on the security administrator or the intruder, they are not included in our analysis.

To avoid confusion, we make a distinction between the notions of response and counter-measure. In the following, we call *counter-measure* any action used as a compensation or a correction. Therefore, a counter-measure changes the system state so that the intrusion is stopped. We define *response as the decision mechanism used to choose the adequate counter-measure when an intrusion is detected.*

Our approach of response mechanism is integrated in the recognition process of the intruder's intentions presented in [3]. When an intrusion scenario is identified, we can anticipate on the objective that the intruder attempts to achieve and on the future attack that he or she will perform to achieve it. Thus, a response is an action that modifies the system state to prevent the intruder to achieve his or her goal. As explained in the following section, a goal can be an intrusion objective or a future attack.

III. MODELLING INTRUSION AND COUNTER-MEASURE

In this section, we present our formalism, based on LAMBDA [7], to model both intrusions and counter-measures.

III.1. Modelling attack and intrusion objective in LAMBDA

LAMBDA is the acronym for LAnguage to Model a dataBase for Detection of Attacks. It is used to provide a logical description of an attack. This description is generic, in the sense that it does not include elements specific to a particular intrusion detection process.

A LAMBDA description of an attack is composed of several attributes:

– **pre-condition** defines the state of the system required for the success of the attack.

– **post-condition** defines the state of the system after the success of the attack.

– **detection** is a description of the expected alert corresponding to the detection of the attack.

– **verification** specifies the conditions to verify the success of the attack[1]. We define an intrusion objective as a specific system state [3]. This state is characteristic of a violation of

1. The alerts launched by IDS generally provide evidence of the occurrence of some malicious events but are not sufficient to conclude that these events will actually cause some damage to the target system. This depends on the system state when the malicious events occur. This is why a LAMBDA description also includes a verification attribute that provides conditions to be checked to conclude that the attack is a success.

the security policy. A LAMBDA description of an intrusion objective is composed by only one attribute: state defines the state of the system that corresponds to a security policy violation.

III.2. How to use LAMBDA

LAMBDA is used to describe possible violations of security policy (intrusion objective) and possible actions (attack) an intruder can perform on a system to achieve an intrusion objective. This database of LAMBDA descriptions is used to recognize an intrusion process and to predict the intention of the intruder.

Let us present an example of intrusion modelled with LAMBDA. First, an intruder scans port 139. If it is open, he concludes that the Operating System is Windows and uses the NetBios service. The intruder can then execute a Winnuke attack on this target system that will cause a denial of service. Figures 1 and 2 respectively give the description in LAMBDA of the Port-Scan and Winnuke attacks performed by an agent on a given host.

attack	$port\_scan(A, H, P)$	
pre:	$open(H, P)$	– port P is open on host H
detection:	$classification(Alert, 'TCP\_Scan')$	– the alert classification is 'TCP_Scan'
	$\wedge\ source(Alert, A)$	– the source in alert is A
	$\wedge\ target(Alert, H)$	– the target in alert is H
	$\wedge\ target\_service\_port(Alert, P)$	– the scanned port is P
post:	$knows(A, open(H, P))$	– agent A knows that port P is open
verification:	$true$	– always true

FIG 1. – Port-Scan Attack performed by an agent *A* on a given host *H*.

Attaque Port-Scan réalisée par un attaquant A sur une machine H.

attack	$winnuke(A, H, S)$	
pre:	$use\_os(H, windows)$	– OS on host H is Windows
	$\wedge\ use\_service(H, 'Netbios')$	– host H uses '$Netbios$' service
	$\wedge\ open(H, 139)$	– port 139 is open on host H
detection:	$classification(Alert, 'Winnuke')$	– alert classification is '$Winnuke$'
	$\wedge\ source(Alert, A)$	– source in alert is A
	$\wedge\ target(Alert, H)$	– target in alert is H
post:	$deny\_of\_service(H)$	– deny of service on H
verification:	$unreachable(H)$	– host H does not reply

FIG 2. – Winnuke Attack performed by an agent *A* on a given host *H*.

Attaque Winnuke réalisée par un attaquant A sur une machine H.

There is a violation of security policy when a web server goes down. This is represented by the intrusion objective presented in Figure 3.

objective	$webserver\_failure(H)$	
state:	$deny\_of\_service(H)$	– deny of service on host H
	$\wedge\ server(H, http)$	– host H is an http server

FIG 3. – Intrusion objective: Denial of service on a web server.

Objectif d'intrusion: déni de service sur un serveur de la toile.

As we can see in these examples, each LAMBDA description uses several variables (corresponding to terms starting with an upper case letter). When an alert can be associated with a LAMBDA description through the *detection* attribute, we can unify variables with values. We call *attack occurrence* a LAMBDA description where variables have been unified with values.

III.3. Using LAMBDA to model counter-measure

We suggest adopting the same formalism to model counter-measures. Thus, a counter-measure has similar attributes to an attack. The main difference is that the *detection* attribute associated with an attack is replaced by the attribute *action.* This leads to the following model for counter-measures:

– **pre-condition** defines the system state required for the success of the counter-measure.
– **post-condition** defines the system state after applying the counter-measure.
– **action** defines the actions necessary to perform the counter-measure.
– **verification** specifies the conditions to verify the success of the counter-measure.

Figure 4 provides an example of a counter-measure specified in this model. It consists in closing all connections between a given source *S* and a given target *T*.

counter-measure	$close\_remote\_access(S, T)$	
pre:	$remote\_access(S, T)$	– S has a remote access to T
action:	$TCP\_reset(S, T)$	– a TCP_reset closes the connection
post:	$not(remote\_access(S, T))$	– connections closed between both side
verification:	$not(TCP\_connection(S, T))$	– verify that all connections are closed

FIG 4. – Counter-measure: Closing a TCP connection.

Contre-mesure: fermeture d'une connexion TCP.

As for the attacks and objectives, the approach is to use this formalism to specify a library of possible counter-measures that apply to the system to block an intrusion. We shall now define a response mechanism to select the adequate counter-measures for a detected scenario. This mechanism is based on a principle called *anti-correlation* which is close to the *correlation* principle suggested in [6]. These two principles are formally presented in the following section.

IV. CORRELATION AND ANTI-CORRELATION

Our response mechanism is based on recognizing the intruder's intentions. Using LAMBDA, [6] shows how to correlate detected attacks to identify a scenario. Section IV.1 recalls the definition for the correlation principle. It is then possible to extrapolate this scenario to predict future attacks that the intruder will probably perform and the objective that he attempts to achieve. When several possible scenarios are extrapolated, [2] suggests an approach to define an order of preference between these scenarios to select the most likely ones.

To design the response process, we suggest a second notion, called *anti-correlation* that is formally defined in Section IV.3 and then used in Section V to select the counter-measure candidates in the response process.

IV.1. Correlation

Our approach of correlation is based on the unification principle [15] on predicates[2]. Let *a* and *b* be two LAMBDA descriptions of attacks. $post_a$ is the set of literals of *post-condition*[3] of *a* and pre_b *is* the set of literals of *pre-condition* of *b*.

Direct correlation: *a* and *b* are directly correlated if the following condition is satisfied:
$\exists\ E_a$ and E_b such that
– $(E_a \in post_a \wedge E_b \in pre_b)$ or $(not(E_a) \in post_a \wedge not(E_b) \in pre_b)$
– and E_a and E_b are unifiable through a most global unifier θ.

This definition of direct correlation represents the idea of positive influence between two attacks. We say that attack *a* has a positive influence over attack *b* if *a* is directly correlated to *b*. In such a case, the effects of *a*, namely the set of predicates in $post_a$, allows to satisfy a subset of the pre-requisites of pre_b. The notion of attack correlation allows us to find correlated attacks that are part of the same scenario.

2. As used in PROLOG.
3. *Post-condition* is represented in its conjunctive form.

[6] also defines the notion of *Knowledge gathering correlation*, a variation of the above definition of correlation that is useful to integrate, in the detection process, preliminary steps the intruder performs to collect data on the target system.

Knowledge gathering correlation: *a* and *b* are knowledge gathering correlated if the following condition is satisfied:

$\exists\, E_a$ and E_b such that

– $(knows\ (Agent,\ E_a) \in post_a \wedge E_b \in pre_b)$ or $(knows\ (Agent,\ not\ (E_a) \in post_a \wedge not(E_b) \in pre_b)$

and E_a and E_b are unifiable through a most global unifier θ.

This definition generally applies to the first steps of an intrusion. We say "generally" because an intruder may have no knowledge about the target machine. An intruder may directly try to exploit a vulnerability on a machine without trying to know if this security hole is present on the machine, but we argue that most of the time, the intruder will try to gather some information about the target. Hence the gathered knowledge may influence the attacker on the next attacks he will execute.

As an example, there is a knowledge gathering correlation between the Port-Scan attack (see Figure 1) and the Winnuke attack (see Figure 2) through the predicate *open* and the unifier that matches variable *H* in both attack definitions and variable *P* in the Port-Scan attack to constant 139. This means that an intruder who knows that port 139 is open on a given host, can then perform a Winnuke attack on this host.

We now define the notion of correlation unifier that allows us to apply on-line correlation.

Correlation unifier: denoted Ξ_{ab}, *is* the set of all possible unifiers[4] to correlate $post_a$ and pre_b.

Since two attacks *a* and *b* are correlated as soon as they have one predicate in common in $post_a$, and pre_b, we may have several unifiers for two attacks. The set of correlation unifiers allows us to know which attack can be correlated with a given attack under some unification condition between their variables. Applying on-line correlation consists in exploring the set of correlation unifiers each time a new alert is received, given that the alert corresponds to an instance of an attack model.

We can apply the notion of direct correlation between two attacks to an intrusion objective and an attack. This allows us to detect that some attack may allow to reach or help to reach an intrusion objective. In this case, we simply have to substitute the term *pre-condition* by *state* in the definition of direct correlation.

IV.2. How to use correlation

Once attacks and intrusion objectives are specified in LAMBDA, we can generate all correlation unifiers between each pair of attacks (respectively between an attack and an intrusion objective). When two attack occurrences are detected, if some unifier in the unifier set is

4. Unifiers of direct or knowledge gathering correlation.

identified, we can then say that these attack occurrences are correlated in the same intrusion scenario.

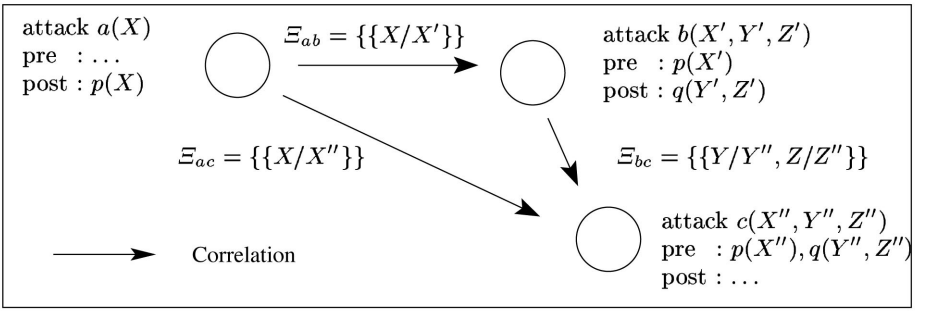

FIG 5. – Correlation graph example.

Exemple de graphe de corrélation.

Using this approach, it is possible to build a correlation graph. Figure 5 presents such a correlation graph where nodes are LAMBDA descriptions and edges are correlation unifiers.

When the first steps of a given intrusion scenario are identified, we can, with the same mechanisms, predict possible continuations of this scenario. We can generate hypothesis about future attacks and the intrusion objectives the intruder attempts to achieve. We shall call *virtual attack* an attack predicted by this process of intention recognition. A virtual attack becomes effective once its occurrence is detected.

Thus, it is sometimes possible to anticipate on the actions performed by the intruder and develop a specific counter-measure in response. This means that our approach may be used to launch a counter-measure not only after a given intrusion objective is achieved by the intruder but also when the beginning of a given scenario is detected. In this latter case, the counter-measure will be used to prevent continuations of this starting scenario.

We shall now see how to define and use the anti-correlation principle to elaborate the counter-measure.

IV.3. Anti-correlation

Let *a* and *b* be respectively LAMBDA descriptions of a counter-measure and an attack. $post_a$ is the set of literals of *post-condition* of *a* and pre_b is the set of literals of *pre-condition* of *b*.

Anti-correlation: *a* and *b* are anti-correlated if the following condition is satisfied:
$\exists\ E_a$ and E_b such that
– $(E_a \in post_a \wedge not(E_b) \in pre_b)$ or $(not(E_a) \in post_a \wedge E_b \in pre_b)$
– and E_a and E_b are nnifiable through a most global unifier θ.

This definition formalizes the notion of negative impact of a counter-measure over an intrusion scenario. A counter-measure is an action which prevents the execution of an attack. Since our model of an attack includes the necessary conditions the system's state must meet in order to execute the attack, we can prevent the execution of an attack by making one of those conditions false. So a counter-measure *c* for an attack *a* is a LAMBDA model of which the post-condition contains a predicate that contradicts one predicate of pre_a. We say that *c* is anti-correlated with *a*. It is sufficient for a counter-measure to anti-correlate an attack through only one predicate, but a counter-mesure can anti-correlate an attack through several predicates.

Anti-correlation unifier: denoted Ψ_{ab}, *is* the set of all unifiers θ possible to anti-correlate $post_a$ and pre_b.

As for a correlation unifier, an anti-correlation unifier defines which attacks can be anti-correlated to a counter-measure. It tells how the variables must be unified in the predicates which are involved in the anti-correlation link.

Using the same approach, it is possible to define anti-correlation between a counter-measure and an intrusion objective. We have simply to replace *pre-condition* by *state* in the previous definition.

In the following section we show how to use the anti-correlation notion to design a response mechanism to an intrusion scenario. In particular, Figures 6 and 7 provide examples of anti-correlation and how to use it in a response mechanism.

V. USING ANTI-CORRELATION FOR RESPONSE

When a scenario is identified, the correlation process provides a graph of attack occurrences, virtual attacks and intrusion objective. A counter-measure will apply to invalidate future attacks or invalidate an intrusion objective. Thus, we have two response mechanisms, one that applies against virtual attacks and the other on an intrusion objective.

V.1. Response to an intrusion objective

In this case, response aims at updating the system state to invalidate the intrusion objective in an intrusion scenario.

Let *o* be an intrusion objective. To invalidate this intrusion objective, we must find a LAMBDA definition *r* of a counter-measure such that $\Psi_{r_0} \neq \theta$. Then, it is possible to parameterize this counter-measure candidate with the unifier of correlation Ψ_{r_0}.

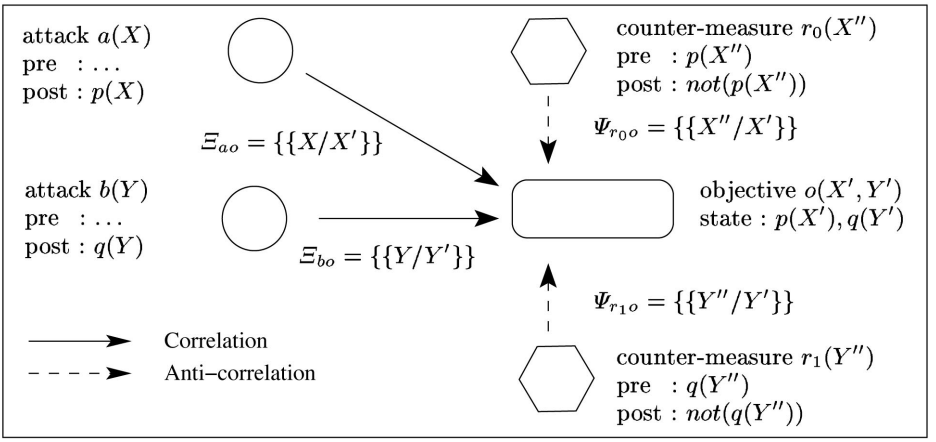

FIG 6. – Correlation graph with direct response on objective.

Graphe de corrélation avec réponse directe sur un objectif.

In Figure 6, we assume that two occurrences of attack are detected: an occurrence of *a* with argument $X = x$ and an occurrence of *b* with argument $Y = y$. The correlation process diagnoses that these two attacks are correlated with a given intrusion objective *o* and this objective is achieved. The response process finds two counter-measure candidates: (1) counter-measure r_0 with parameter $X'' = X'$, provided by $\Psi_{r_o,o}$, to invalidate condition *p* (and thus objective *o*) and (2) counter-measure r_1 with parameter $Y'' = Y'$, provided by $\Psi_{r_1,o}$, that invalidates condition *q* (and thus also objective *o*). Combining $\Xi_{a,o}$ with $\Psi_{r_o,o}$ we can derive that counter-measure r_0 may apply with parameter $X'' = X' = X = x$ and similarly counter-measure r_1 may apply with parameter $Y'' = y'$. Thus this provides means to derive which parameters must be selected when applying the counter-measure. In our approach, these two counter-measures are suggested to the administrator who can select one of them (or both). The *verification* field of the selected counter-measure is then evaluated to check if the counter-measure was executed successfully. If this is the case, we can reevaluate the state condition of the intrusion objective to false.

V.2. Response to an ongoing scenario

It is possible that a counter-measure may not apply directly to an intrusion objective if one of these conditions holds:

– There is not any counter-measure in the response library which may apply to invalidate the intrusion objective.

– The counter-measure does not apply to the system state because the pre-condition of this counter-measure is evaluated to false.

– All counter-measure candidates were launched without success.

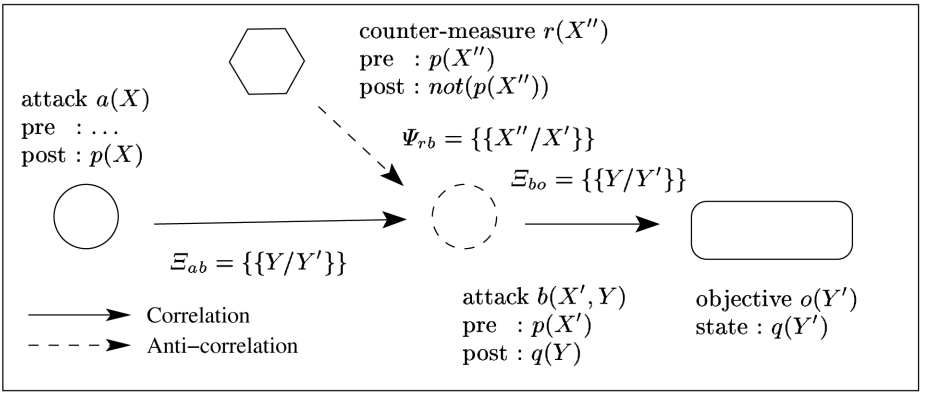

FIG 7. – Correlation graph with response on a sequence of virtual attacks.

Graphe de corrélation avec réponse sur une séquence d'attaques virtuelles.

In these cases, a possible solution is to modify the system state to invalidate one attack in a sequence of virtual attacks. When the correlation engine receives a new alert, it tries to find a path of correlated virtual attacks leading to one (or more) intrusion objective(s). The path of virtual attacks and the intrusion objective found represents a possible evolution of the ongoing scenario. Let $a_1 ... a_n$ be a sequence of virtual attacks and o an intrusion objective such that for every $i \in [1, n-1]$, a_i is correlated with a_{i+1} and a_n is correlated with o. The affirmation stating that for every $i \in [1, n-1]$, a_i is correlated with a_{i+1} and a_n is correlated with o is not necessarily true for the entire set of virtual attack generated. But we can always find a subset of virtual attack satisfying this condition in the set of generated virtual attacks, given that the set of generated virtual attacks leads to an intrusion objective.

To block this sequence of attacks, we must find a valid LAMBDA counter-measure r such that r is anti-correlated with one of the attacks a_k ($k \in [1, n]$).

For instance, let us assume, in Figure 7, that we detect an occurrence of a. The recognizing intention process identifies that the intruder may perform b after a to achieve the objec-

tive *o*. In this case, the response process can find a counter-measure *r* to invalidate the *pre-condition* of *b.* This will prevent performance of attack *b* and invalidate this scenario.

VI. EXAMPLES AND EXPERIMENTATION

This section presents the Prevention Cells framework [8], a cooperative intrusion detection platform that implements the response mechanism suggested in this paper. This framework is a research prototype implemented in C and C++, and has been tested on different versions of Linux 2.4.x series and on the versions 2.9.x and 3.x of GNU's gcc compiler. The global architecture of this platform is shown in Figure 8.

The platform works as follows. First, the event watcher component (ewatcher-ids) collects the set of events raised by different host and network based sensors and translates them into IDMEF (Intrusion Detection Message Exchange Format). These alerts, together with the IDMEF alerts collected by other third party IDS included in our platform (such as snort and prelude) are sent to an alert database (managed by PostgresSQL). An enhanced C++ version

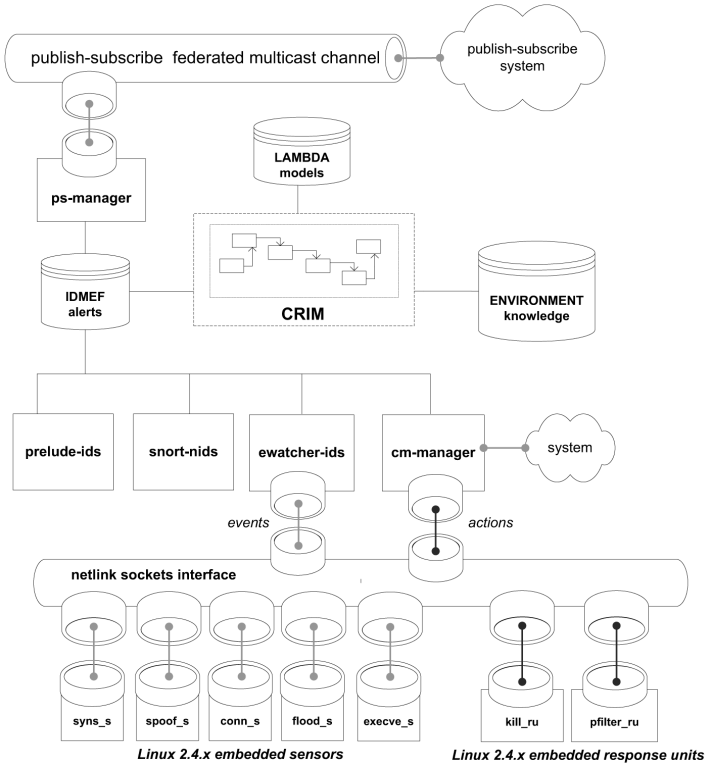

FIG 8. – Cooperative intrusion detection platform.

Plateforme de détection d'intrusion coopérative.

of CRIM [6] analyzes these IDMEF alerts using the approach presented in Section IV.1. To generate the corresponding counter-measures, CRIM uses the approach based on anti-correlation defined in Section V. These counter-measures are transmitted to the countermeasure manager component (cm-manager) as IDMEF assessment alerts. Finally, the counter-measure manager provides the administrator with a set of actions. The administrator in charge can then select one or several actions that automatically will be executed at the corresponding response unit.

The response units associated with the counter-measure manager, as well as the sensors associated to the ewatcher-ids component, are implemented in C and executed as linux 2.4 modules. To illustrate the use of two of these response units let us consider two attack scenarios: an illegal remote command based on the *Mitnick attack* and a distributed denial of service attack.

VI.1. Mitnick attack

This attack tries to exploit the trust relationship between two computers to achieve an illegal remote access using the coordination of three techniques. First, a SYN flooding DoS attack to keep the trusted system from being able to transmit. Second, a TCP sequence prediction against the target system to obtain its following TCP sequence numbers. And third, an

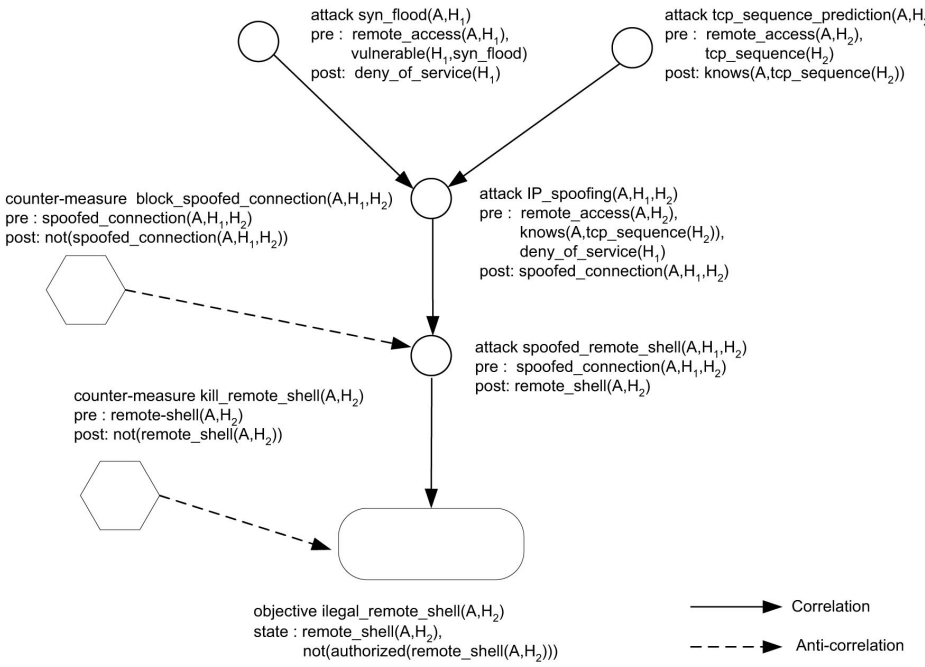

FIG 9. – Correlation graph for the Mitnick attack scenario.

Graphe de corrélation pour le scénario d'attaque Mitnick.

unauthorized remote shell by spoofing the IP address of the trusted system (while it is in a mute state) and using the sequence number that the target system is expecting. The correlation graph for this attack is presented in Figure 9.

The LAMBDA models for each attack that composes the whole scenario, together with the intrusion objective, are shown in Figure 10. In the first step, A (the agent that performs the whole attack) floods a given host H_1. In the second step, A sends a TCP sequence prediction attack against host H_2 to obtain its following TCP sequence numbers. Then, by using these TCP sequence numbers, A starts a spoofed remote shell session to the host H_2 as it would come from host H_1. Since H_1 is in a mute state, H_2 will not receive the RST packet to close this connection. If the third and fourth step are successful, A will establish an illegal remote shell session to system H_2.

attack	$syn\_flood(A, H)$	
pre:	$remote\_access(A, H)$	– attacker has a remote access on the target
	$\wedge$ $vulnerable(H,'syn\_flood')$	– target is vulnerable to 'syn_flood' attack
detection:	$classification(Alert,'syn\_flood')$	– the alert classification is 'syn_flood'
	$\wedge$ $source(Alert, A)$	– the source in alert is agent A
	$\wedge$ $target(Alert, H)$	– the target in alert is host H
post:	$deny\_of\_service(H)$	– deny of service on host H
verification:	$unreachable(H)$	– host H does not reply
attack	$tcp\_sequence\_prediction(A, H)$	
pre:	$remote\_access(A, H)$	– attacker has a remote access on the target
	$\wedge$ $tcp\_sequence(H)$	– the TCP sequence of H is predictable
detection:	$classification(Alert,'TCP\_seq\_prediction')$	– the alert classification is '$TCP\_seq\_prediction$'
	$\wedge$ $source(Alert, A)$	– the source in alert is agent A
	$\wedge$ $target(Alert, H)$	– the target in alert is host H
post:	$knows(A, tcp\_sequence(H))$	– attacker knows the TCP sequence of target
verification:	$true$	– always true
attack	$IP\_spoofing(A, H_1, H_2)$	
pre:	$remote\_access(A, H_2)$	– attacker has a remote access on H_2
	$\wedge$ $knows(A, tcp\_sequence(H_2))$	– attacker knows the TCP sequence of H_2
	$\wedge$ $deny\_of\_service(H_1)$	– deny of service on host H_1
detection:	$classification(Alert,'IP\_spoofing')$	– the alert classification is '$IP\_spoofing$'
	$\wedge$ $source(Alert, A)$	– the source in alert is agent A
	$\wedge$ $source(Alert, spoofed)$	– the source is spoofed
	$\wedge$ $additional\_data(Alert, spoofed\_addr, H_1)$	– the spoofed address is H_1
	$\wedge$ $target(Alert, H_2)$	– the target in alert is host H_2
post:	$spoofed\_connection(A, H_1, H_2)$	– attacker has a spoofed connection on H_2 as H_1
verification:	$unreachable(H_1)$	– host H_1 does not reply
counter-measure	$block\_spoofed\_connection(A, H_1, H_2)$	
pre:	$spoofed\_connection(A, H_1, H_2)$	– attacker has a spoofed connection on H_2 as H_1
action:	$block\_IP\_datagrams(A, H_1, H_2, t)$	– a packet filtering blocks the connection
post:	$not(spoofed\_connection(A, H_1, H_2))$	– spoofed connections blocked between both side
verification:	$not(TCP\_spoofed\_connection(A, H_1, H_2))$	– verify that spoofed connections are blocked
attack	$spoofed\_remote\_shell(A, H_1, H_2)$	
pre:	$spoofed\_connection(A, H_1, H_2)$	– attacker has a spoofed connection on H_2 as H_1
detection:	$classification(Alert,'spoofed\_rshell')$	– the alert classification is '$spoofed\_rshell$'
	$\wedge$ $source(Alert, A)$	– the source in alert is agent A
	$\wedge$ $source(Alert, spoofed)$	– the source is spoofed
	$\wedge$ $additional\_data(Alert, spoofed\_addr, H_1)$	– the spoofed address is H_1
	$\wedge$ $target(Alert, H_2)$	– the target in alert is host H_2
post:	$remote\_shell(A, H_2)$	– attacker has a remote shell on Host H_2
verification:	$unreachable(H_1)$	– host H_1 does not reply
counter-measure	$kill\_remote\_shell(A, H)$	
pre:	$remote\_shell(A, H)$	– attacker has a remote shell on the target
action:	$kill\_process(remote\_shell(A, H))$	– a system call kills the corresponding process
post:	$not(remote\_shell(A, H))$	– remote shell between A and H_2 is closed
verification:	$not(process\_running('rsh', A, H)$	– verify that the remote shell process is closed
objective	$illegal\_remote\_shell(A, H)$	
state:	$remote\_shell(A, H)$	– attacker has a remote shell on host H
	$\wedge$ $not(authorized(remote\_shell(A, H)))$	– attacker is not authorized to have this remote shell

FIG 10. – Modelling the Mitnick attack scenario.

Modélisation du scénario d'attaque Mitnick.

When alerts corresponding to different steps of this scenario are raised, CRIM applies the correlation mechanism recalled in this paper to recognize the global scenario. For example, if an alert corresponding to the attack *syn_flood*(*A*, H_1) and the attack *tcp_sequence_prediction*(*A*, H_2) are raised, the correlation mechanism of CRIM will generate the virtual alerts corresponding to the attack *IP_spoofing*(*A*, H_1, H_2), the attack *spoofed_remote_shell*(*A*, H_1, H_2) and the objective *illegal_remote_shell*(*A*, H_2). CRIM recognizes then the whole scenario since the post-condition *deny_of_service* (H_1) of the attack *syn_flood*(*A*, H_1) *is* correlated with the pre-condition *deny_of_service* (H_1) of the attack *IP_spoofing*(*A*, H_1, H_2).

Likewise, the pre-condition *spoofed_connection*(*A*, H_1, H_2) of the attack *spoof ed_remote_shell*(*A*, H_1, H_2) *is* correlated with the post-condition *spoofed-connection*(*A*, H_1, H_2) *of IP_spoofing*(*A*, H_1, H_2), the pre-condition *knows*(*A*, *tcp_sequence*(H_2)) *of IPspoofing*(*A*, H_1, H_2) *is* correlated with the post-condition *knows*(*A*, *tcp_sequence*(H_2)) of *tcp_sequence_prediction*(*A*, H_2), and the intrusion objective state *remote_shell* (*A*, H_2) is then correlated with the post-condition *remote_shell* (*A*, H_2) of the attack *spoofed_ remote_shell*(*A*, H_1, H_2).

Once this diagnosis is obtained, the response module is used to select some counter-measures in the counter-measure library. Faced to this intrusion scenario, the response module can select two possible counter-measures. The LAMBDA models for these two counter-measures are also shown in Figure 10. On one hand, a counter-measure can apply directly on the intrusion objective. This counter-measure, called *kill_remote_shell*, kills the remote shell process. On the other hand, we can react before the intrusion objective is achieved. The counter-measure, called *block_spoofed_connection*, blocks the spoofed connection between the host and the intruder. The counter-measure *block-spoofed-connection* is chosen since its post-condition *not* (*spoofed_connection* (*A*, H_1, H_2)) is anti-correlated with the pre-condition *spoof ed_connection*(*A*, H_1, H_2) from *spoofed_remote_shell* (*A*, H_1, H_2). Likewise, the counter-measure *kill_remote_shell* is chosen since its predicate *not*(*remote_shell*(*A*, H_2)) is then anti-correlated with the predicate *remote_shell* (*A*, H_2) of the intrusion objective *illegal_remote_shell* (*A*, H_2).

These two counter-measures are transmitted to the counter-measure manager as IDMEF assessment alerts, which provides the administrator with the actions showed in Figure 10. Thus, the administrator can select one of them, or both. For example, if the administrator chooses the first action, *block_IP_datagrams*(*A*, H_1, H_2, *t*), the counter-measure manager will send this information to the response unit *pfilter_ru*. The implementation of this response unit is based on the netfilter subsystem, a framework for packet manipulation that enables packet filtering, network address translation and other packet mangling on Linux 2.4.x and upper series. When the response unit *pfilter_ru* receives the information sent by the counter-measure manager, it produces the corresponding packet filtering on netfilter to block the spoofed IP datagrams during the timeout *t*.

On the other hand, if the administrator chooses the second action, *kill_ _process*(*remote_shell*(*A*, H_2)), the counter-measure manager will send this information to the response unit *kill_ru*. The implementation of this response unit is based on the interception of the *kill* system call to provide a safe mechanism to invoke the original *kill* system call using the Netlink Sockets API. It allows us to find and to terminate the execution of the processes reported by the counter-measure manager (in that case, the execution of the *remote-shell* process to the host H_2).

After the execution of the chosen action, the counter-measure manager receives the acknowledgment from the corresponding response unit. Then, it will send an IDMEF alert to CRIM

to tell that the given counter-measure has been launched. Finally, CRIM can apply the *verification* field of the counter-measure to check if the counter-measure has been effective.

VI.2. Distributed Denial of Service

In this section, we present how to use the alert correlation discussed in section IV.1 to detect an ongoing DDoS attack scenario and how to stop it using anti-correlation (section IV.3). Denial of Service (DoS) attacks, based on flooding, consist in putting out of service some physical or logical resource in a computer system. The attacker sends huge amounts of legitimate traffic in order to decrease, or stop a specific service on the victim machine. The excessive consumption of the victim's resources deprives legitimate clients from the offered victims' service for many minutes or hours as long as the attack is active.

Distributed Denial of Service attacks, based on flooding, use the architecture shown in Figure 11 where the attacker recruits many agents known as zombies to flood the designed victim(s). The different hosts which are between the attacker and the zombies are called masters (or handlers). These hosts are servers, which are recruited by the attacker(s), offering many commands facilities to launch the desired flooding attacks.

The LAMBDA model corresponding to the global scenario is shown in Figure 12. We mention that we do not take into account the early steps of the DDoS tools that consist in scanning and compromising the different hosts that will later play the role of masters and slaves. These intrusions may be detected with some IDSs and then some corresponding scenarios

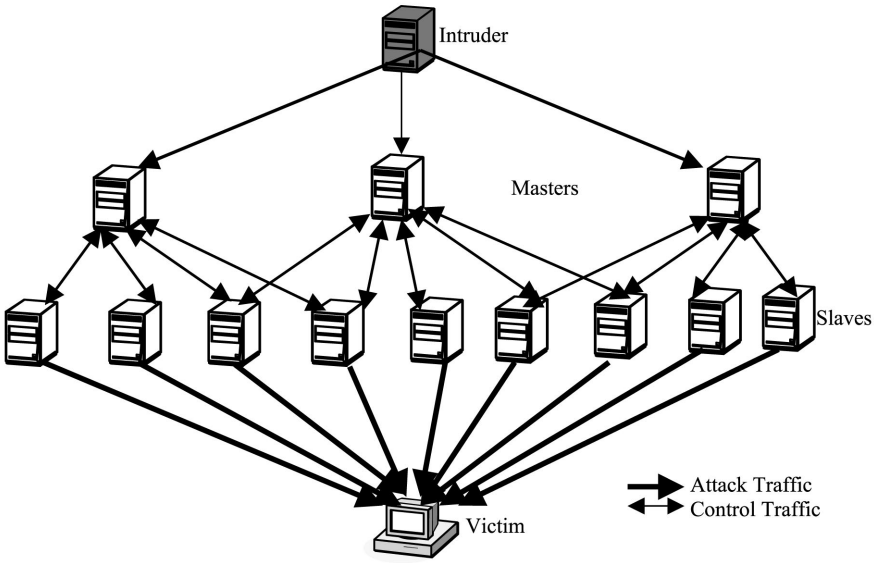

FIG 11. – DDoS Attack Architecture.

Architecture d'une attaque DDoS.

may be constructed which will be detected with the correlation method discussed above. However, if the attacker has a legitimate physical or remote access to the different machines then these first steps cannot be detected.

The scenario shown in Figure 12 corresponds to the different steps followed by an attacker, after compromising the different useful hosts, to launch a distributed denial of service against victims. In reality, it corresponds to the scenario of activating well known DDoS tools such as Trinoo, Stacheldraht, TFN, Mstream, Shaft, etc. Once activated on the compromised hosts, these DDoS tools perform classical DoS attacks such as syn-flooding or smurfing.

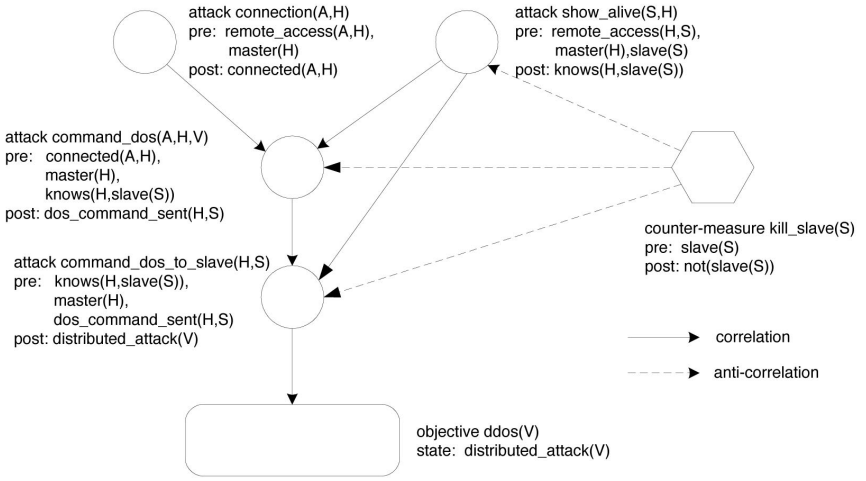

FIG 12. – DDoS Attack correlation graph.

Graphe de corrélation d'une Attaque DDoS.

The LAMBDA models for each elementary attack corresponding to the scenario is shown in Figure 13.

The first step consists in opening a connection between the attacker with the master. The pre-condition of the first attack of the scenario mentions that the H host is a master host compromised by the attacker A. Its post-condition specifies that the attacker has opened a connection with the master.

When the attacker launches (automatically or not) the daemon(s) on the different compromised slaves hosts, these slaves hosts will send a message called "*show alive message*" to the masters that control them in order to inform their readiness to flood a victim. This action may be performed in parallel with the first step.

In the third step, the attacker sends a *dos* command to the master to start a DDoS attack using the detected slave computer(s). Thereafter, the master sends a DoS command to the slave in order to flood the desired victim(s).

attack	$connection(A, H)$	
pre:	$remote\_access(A, H)$	– attacker has a remote access on H
	$\wedge\ master(H)$	– H is a DDoS master host H
detection:	$classification(Alert,'master')$	– the alert classification is '$master$'
	$\wedge\ source(Alert, A)$	– the source in alert is agent A
	$\wedge\ target(Alert, H)$	– the target in alert is host H
post:	$connected(A, H)$	– attacker has *opened* a connection to the master H
verification:	$true$	– always true
attack	$show\_alive(S, H)$	
pre:	$remote\_access(S, H)$	– The slave S has a remote access on the master H
	$\wedge\ master(H)$	– H is a DDoS master host
	$\wedge\ slave(S)$	– S is a DDoS slave host
detection:	$classification(Alert,'show\_alive(S, H)')$	– the alert classification is '$show\_alive(S, H)$'
	$\wedge\ source(Alert, S)$	– the source in alert is slave S
	$\wedge\ target(Alert, H)$	– the target in alert is master host H
post:	$knows(H, slave(S))$	– master H knows that host S is an alive slave
verification:	$true$	– always true
counter-measure	$kill\_slave(S)$	
pre:	$slave(S)$	– H is a DDoS slave host
action:	$kill\_slave(S)$	– kill the daemon of the running slave process
post:	$not(slave(S))$	– H is not a DDoS slave host
verification:	$not(slave(S))$	– verify that there is no slave running on host S
attack	$command\_dos(A, H, V)$	
pre:	$connected(A, H)$	– attacker has *opened* a connection to the master H
	$\wedge\ master(H)$	– H is a DDoS master host
	$\wedge\ knows(H, slave(S))$	– master H knows that host S is an alive slave
detection:	$classification(Alert,'command\_dos')$	– the alert classification is '$command\_dos$'
	$\wedge\ source(Alert, A)$	– the source in alert is agent A
	$\wedge\ additional\_data(Alert,'ddos\_victim', V)$	– the victim to flood is V
	$\wedge\ target(Alert, H)$	– the target in alert is host H
post:	$dos\_command\_sent(H, S, V)$	– master H sends a dos command to slave S to flood victim V
verification:	$true$	– always true
attack	$command\_dos\_to\_slave(H, S, V)$	
pre:	$knows(H, slave(S))$	– master H knows that host S is an alive slave
	$\wedge\ master(H)$	– H is a $DDoS$ master host
	$\wedge\ dos\_command\_sent(H, S, V)$	– master H sends a dos command to slave S
detection:	$classification(Alert,'command\_dos\_to\_slave')$	– the alert classification is '$command\_dos\_to\_slave$'
	$\wedge\ source(Alert, H)$	– the source in alert is Master host H
	$\wedge\ additional\_data(Alert,'ddos\_victim', V)$	– the victim to flood is V
	$\wedge\ target(Alert, S)$	– the target in alert is agent host S
post:	$distributed\_denial\_of\_service(V)$	– distributed denial of service on V
verification:	$unreachable(V)$	– host V does not reply
objective	$ddos(V)$	
state	$distributed\_denial\_of\_service(V)$	– distributed denial of service on V

FIG 13. – Modelling a DDoS scenario.

Modélisation d'un scénario DDoS.

Once the CRIM engine has recognized the global scenario using the approach explained in the previous sections, one appropriate counter-measure should be launched. It consists in killing the slave daemon process. In addition to this, the administrator where the daemon is located is warned about the host where the daemon is installed. He should find the vulnerability that permitted the installation of the daemon on that host and disinfect and patch the system. Without doing this, other daemons may be launched automatically from the same host. This is what is called counter counter-measure.

We mention that the main goal in using correlation technique to recognize the first steps of an ongoing DDoS scenario is to react against it before the objective is reached i.e. before the flooding is started where in this case it is difficult and too late to react.

VII. CONCLUSIONS

In this paper, we have presented a global approach to select and apply response mechanisms when an intrusion occurs. This approach is based on a logical representation in LAMBDA of both intrusions and counter-measures. This is used to build libraries of intrusions and counter-measures.

The library of counter-measure is organized into a taxonomy that takes its inspiration from [9]. The notion of anti-correlation is then used to select relevant responses to a given intrusion in order to help the administrator to decide which appropriate counter-measures may be launched. This mechanism is integrated on an experimental platform of intrusion detection that collects and analyzes alerts generated by various intrusion detection systems. This platform is a research prototype implemented in C and C++.

The response units of our platform are implemented as Linux 2.4 modules and can interact with the other components via the Netlink Sockets API available for GNU/Linux systems.

Up to now, we only use this approach to provide a support to the administrator who takes the final decision to choose and launch a given response. This is a prudent strategy but it introduces an overhead that is sometimes incompatible with real time response. This is why we are currently analyzing situations where it would be possible to automatically decide to launch the response.

Notice that a possible response consists in reconfiguring the security policy to prevent a new occurrence of a given intrusion. However, as suggested in [13], dynamic changes of the security policy may cause failure of some software components. This is why [13] suggests the notion of security agility, a strategy to provide software components with adaptability to security policy changes. Security agility might be nicely included into the intrusion detection and response framework suggested in this paper. This represents a possible extension of our work.

When using anti-correlation, several responses may be selected. In this case, it would be interesting to rank these different responses and a possible ranking criteria would be to evaluate the effectiveness of the responses to stop the attack. For this purpose, we plan to extend the response formalism with temporal logic to include the fact that a given response will stop an intrusion until another additional event occurs. More difficult is performing an action that will cause this additional event, more effective is the response. This also represents further work that remains to be done.

Acknowledgements

This work was supported by funding from the French ministry of research, under the ACI Daddi project, the Spanish Government project TIC2003-02041, and the Catalan Government grants 2003FI126 and 2005BE77

REFERENCES

[1] BACE (R.), *Intrusion Detection*, McMillan, 2000.

[2] BENFERHAT (S.), AUTREL (F.), CUPPENS (F.), Enhanced correlation in a intrusion detection process, In *Mathematical Methods, Models and Architectures for Computer Network Security (MMM-ACNS* 2003), St Petersburg, Russia, September 2003.

[3] CUPPENS (F.), AUTREL (F.), MIÈGE (A.), BENFERHAT (S.), Recognizing malicious intention in an intrusion detection process, Special session: "Hybrid Intelligent Systems for Intrusion Detection", In *Second International Conference on Hybrid Intelligent Systems (HIS'2002)*, vol. 87, pp. 806-817, Santiago, Chile, October 2002.

[4] CUPPENS (F.), Managing alerts in a cooperative intrusion detection environnement, In *17th Annual Computer Security Applications Conference* (ACSAC), pp. 22-32, New Orleans, Louisiana, December 2001.

[5] CUPPENS (F.), GOMBAULT (S.), SANS (T.), Selecting Appropriate Counter-Measures in an Intrusion Detection Framework, In *Third Conference on Security and Network Architectures (SAR'04*), La Londe, France, June 2004.

[6] CUPPENS (F.), MIÈGE (A.), Alert correlation in a cooperative intrusion detection framework, In *Proceedings* of *the IEEE Symposium on research in Security and Privacy*, pp. 202-215 Oakland, USA, May 2002.

[7] CUPPENS (F.), ORTALO (R.), LAMBDA: A language to model a database for detection of attacks, In *Third International Workshop on the Recent Advances in Intrusion Detection (RAID'2000)*, number 1907 in LNCS, pp. 197-216, Toulouse, France, October 2000.

[8] GARCIA (J.), AUTREL (F.), BORRELL (J.), CASTILLO (S.), CUPPENS (F.), NAVARRO (G.), Decentralized publish-subscribe system to prevent coordinated attacks via alert correlation, In *Proceedings of the Sixth International Conference on Information and Communications Security (ICICS'2004)*, number 3269 in LNCS, October 2004.

[9] GOMBAULT(S.), DIOP (M.), Response function, In *First Symposium on Real Time Intrusion Detection (NATO)*, Lisbon, Portugal, May 2002.

[10] JULISCH (K.), Mining alarm clusters to improve alarm handling efficiency, In *17th Annual Computer Security Applications Conference* (ACSAC), New-Orleans, LA, December 2001.

[11] NING (P.), XU (D.), Learning attack strategies from intrusion alerts, In *10th ACM Conference on Computer and Communication Security*, pp. 200-209, Washington DC, USA, 2003.

[12] NORTHCUTT (S.), NOVAK (J.), *Network Intrusion Detection*, Paperback, 1999.

[13] PETKAC (M.), BADGER (L.), Security agility in response to intrusion detection, In *16th Annual Computer Security Applications Conference (ACSAC)*, New-Orleans, LA, 2000.

[14] ROESCH (M.), Short – lightweight intrusion detection for networks, In *Proceedings of LISA'99: 13th Systems Administration Conference*, Seattle, Washington, USA, November 7-12 1999.

[15] DE SWART (H.C.M), *Mathematics, Language, Computer Science and Philosophy*, vol. 2, Peter Lang, 1994.