Selecting Appropriate Counter-Measures in an Intrusion Detection Framework

F. Cuppens, S. Combault, T. Sans
GET/ENST-Bretagne
2, rue de la Châtaigneraie
35576 Cesson Sévigné
France

{frederic.cuppens, sylvain.combault, thierry.sans}@enst-bretagne.fr

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, called DIAMS, that implements the response mechanisms presented in this paper.

Keywords : Intrusion detection, IDMEF, response, counter-measures, correlation, anti-correlation.

1 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, 10, 12]. 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, 8] 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 [5, 9]. 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 acting 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 responses 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.

Paper presented at the RTO IST Symposium on "Adaptive Defence in Unclassified Networks", held in Toulouse, France, 19 - 20 April 2004, and published in RTO-MP-IST-041.

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 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 2 presents the notion of response and suggests several types of response. In section 3, we present our formalism to model attacks and counter-measures. Our formalism is based on LAMBDA, a language suggested in [6] to model attacks. In this section, we also suggest using LAMBDA to model counter-measures. Section 4 recalls the definition of correlation [5] and introduces the notion of anti-correlation. In section 5, 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 6 gives an example to illustrate the approach and presents DIAMS, the platform of intrusion detection that includes the response mechanisms we have presented in this paper. Finally, section 7 concludes the paper and suggests several possible extensions to our approach.

2 Response mechanism and counter-measure

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 [7], 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 miss-configuration 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 prevent 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 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 attempt 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.

3 Modelling intrusion and counter-measure

In this section, we present our formalism, based on LAMBDA [6], to model both intrusions and counter-measures.

3.1 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 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.

3.2 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 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 Scan and Winnuke attacks performed by an *Agent* on a given *Host*.

attack	scan-port(Agent,Host,Port)	
pre:	open(Host,Port)	– Port is open on Host
detection:	classification(Alert,'TCP-Scan')	– the alert classification is 'TCP-Scan'
	$\wedge$ source(Alert,Agent)	– the source in alert is Agent
	$\wedge$ target(Alert,Target)	– the target in alert is Host
	$\wedge$ target_service_port(Alert,Port)	– the scanned port is Port
post:	knows(Agent,open(Host,Port))	– Agent knows that Port is open
verification:	true	– always true

Figure 1: *Scan* Attack performed by an *Agent* on a given *Host*

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

[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.

attack	winnuke(Agent,Host,Service)	
pre:	use_os(Host,windows)	– OS on Host is Windows
	∧ use_service(Host,'Netbios')	– Host uses Netbios service
	∧ open(Host,139)	– Port 139 is open on Host
detection:	classification(Alert,'Winnuke')	– alert classification is winnuke
	∧ source(Alert,Agent)	– source in alert is Agent
	∧ target(Alert,Host)	– target in alert is Host
post:	deny_of_service(Host)	– deny of service on Host
verification:	unreachable(Host)	– Host does not reply

Figure 2: *Winnuke* Attack performed by an *Agent* on a given *Host*

objective	webserver_failure(Host)	
State:	deny_of_service(Host)	– deny of service on Host
	∧ server(Host,http)	– Host is an http server

Figure 3: Intrusion objective: Denial of service on a web server

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, then we can unify variables with values. We call *attack occurrence* a LAMBDA description where variables have been unified with values.

3.3 Using LAMBDA to model counter-measure

We suggest adopting the same formalism to model counter-measure. Thus, a counter-measure have similar attributes as 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 counter-measure specified in this model. It consists in closing a connection between a given *Source* and a given *Target* connected through a *Port-Source* and a *Port-Target.*

counter-measure	close-remote-access(Source,Target)	
pre:	remote-access(Source,Target)	– Source has a remote access to Target
action:	TCP-Reset(Source,Target)	– A TCP-Reset closes the connection
post:	not(remote-access(Source,Target))	
		– Connection closed between Source and Target
verification:	not(TCP-Connection(Source,Port-Source,Target,Port-Target))	
		– Verify that the connection is closed on Target

Figure 4: Counter-measure: Closing a TCP connection

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 [5]. These two principles are formally presented in the following section.

4 Correlation and anti-correlation

Our response mechanism is based on recognizing the intruder's intentions. Using LAMBDA, [5] shows how to correlate detected attacks to identify a scenario. Section 4.1 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 4.3 and then used in section 5 to select the counter-measure candidates in the response process.

4.1 Correlation

Our approach of correlation is based on the unification principle [13] 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
$(E_a \in post_a \wedge E_b \in post_b)$ 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.

[5] 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. This notion is defined as follows:

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

$\exists\ E_a$ and E_b such that
$(knows(Agent, E_a) \in post_a \wedge E_b \in post_b)$
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.

As an example, there is a knowledge gathering correlation between the Scan attack (see figure 1) and the Winnuke attack (see figure 2) through the predicate *open* and the unifier that matches variable *Host* in both attack definitions and variable *Port* in the 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.

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

With the same approach, it is possible to define correlation between an attack and an intrusion objective. In this case, we have simply to replace *pre-condition* by *state* in the previous definition.

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

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 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

[2] as used in PROLOG
[3] *post-condition* is represented in its conjunctive form
[4] Unifiers of direct or knowledge gathering correlation

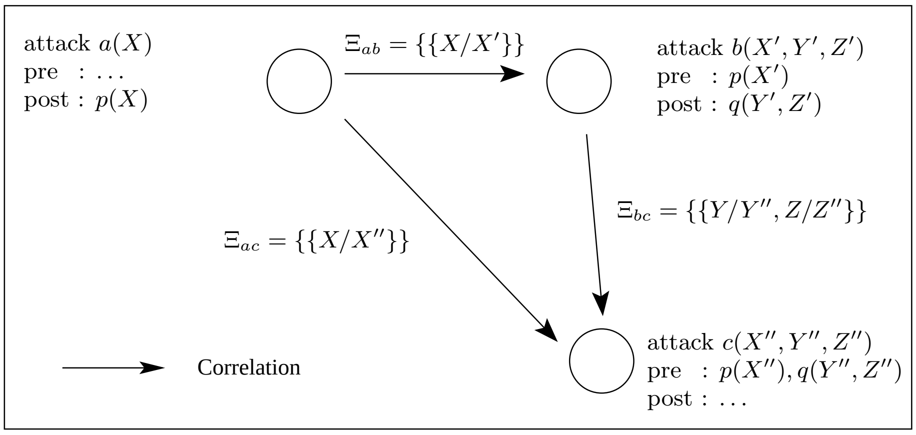
attack $a(X)$
pre : ...
post : $p(X)$
$\Xi_{ab} = \{\{X/X'\}\}$
attack $b(X', Y', Z')$
pre : $p(X')$
post : $q(Y', Z')$
$\Xi_{bc} = \{\{Y/Y'', Z/Z''\}\}$
$\Xi_{ac} = \{\{X/X''\}\}$
Correlation
attack $c(X'', Y'', Z'')$
pre : $p(X''), q(Y'', Z'')$
post : ...

Figure 5: Correlation graph example

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.

4.3 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
$(E_a \in post_a \wedge not(E_b) \in post_b)$ or $(not(E_a) \in post_a \wedge E_b \in pre_b)$
and E_a and E_b are unifiable through a most global unifier θ.

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

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 5, 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.

5 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 intrusion objective. Thus, we have two response mechanisms, one that applies against virtual attacks and the other on an intrusion objective.

5.1 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 $\Psi_{ro} \neq \emptyset$. Then, it is possible to parameterize this counter-measure candidate with the unifier of correlation Ψ_{ro}.

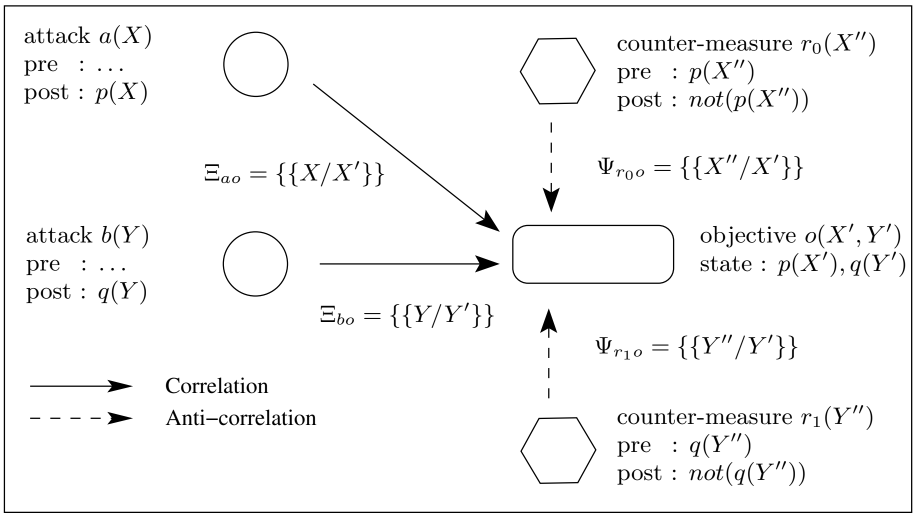

Figure 6: correlation graph with direct response on objective

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 $X'' = X'$ to invalidates condition p (and thus objective o) and (2) counter-measure r_1 with parameter $Y'' = Y'$ that invalidates condition q (and thus also objective o). 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.

5.2 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.

In this case, a possible solution is to modify the system state to invalidate one attack in a sequence of virtual attacks.

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.

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 objective 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.

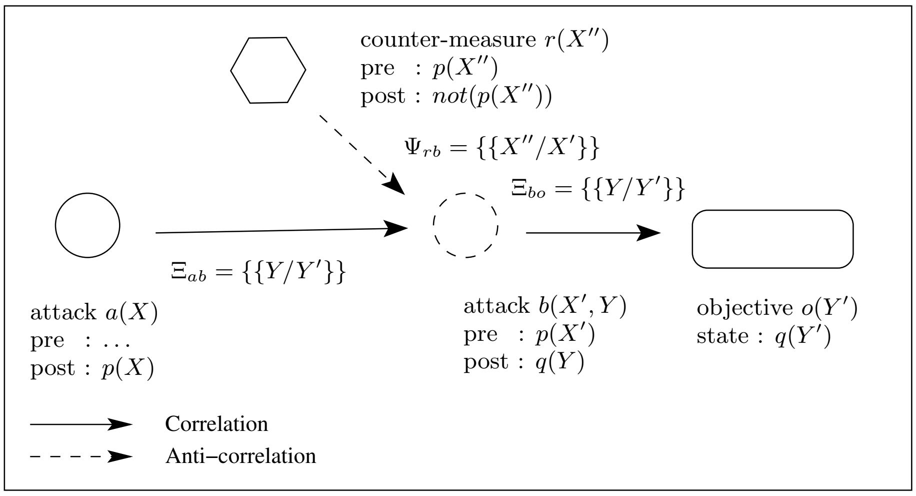

Figure 7: Correlation graph with response on a sequence of virtual attacks

6 Example and experimentation

This section presents DIAMS, a platform of intrusion detection we have developed and implemented in Java. In particular, this platform implements the response mechanism suggested in this paper.

The global architecture of DIAMS is presented in figure 8. DIAMS first collects the syslog alerts raised by different IDS and translates them in the IDMEF (Intrusion Detection Message Exchange Format). Then, these alerts are sent by the router to an alert database (managed by PostgresSQL) and to CRIM. CRIM [5] analyzes these alerts to correlate them using the approach presented in section 4.1 and generates a diagnosis of the detected intrusions. This diagnosis is transmitted to the response mechanism that provides the administrator with a set of counter-measure candidates using the approach based on anti-correlation defined in section 5. The administrator can then select one or several counter-measures. Finally, a module called DIAMS-Action automatically executes the script corresponding to these selected counter-measures.

The CRIM and response modules are currently implemented in Prolog. To illustrate this implementation, let us consider the attack scenario that corresponds to the correlation graph presented in figure 9. This intrusion exploits a system miss-configuration of UNIX export partitions. Through a user partition, the intruder can increase his privilege to get a remote user access on the target host.

With the $rpcinfo$ attack, A (the intruder) collects information about RPC service (Remote Procedure Call) on H (Host). The attack pre-condition specifies that the intruder must have a remote access on host H and this host must have RPC service running. The post-condition specifies that the intruder knows that RPC is running on H.

With the *showmount* attack, an intruder A collects information about export partitions on host H. The pre-condition specifies that H has an export partition P. Then, post-condition specifies that the intruder knows it.

With the *mount* attack, A mounts the partition P from H to his/her local host. The post-condition specifies that the partition is mounted by the intruder in his or her local host.

With the $rhost\text{-}modification$ attack, A modifies the .rhost file whose owner is U in the partition P. If a user account U is hosted by the partition, then the user can get a user access on this account.

With the $login$ attack, A is connected to H through the user account U.

The intrusion objective *illegal-user-access* specifies that the intruder bypasses the password verification and obtains a illegal user access.

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.

Once this diagnosis is obtained, the response module is used to select a counter-measure in the counter-measure library. Faced to this intrusion scenario, the response module actually selects two possible counter-measures. On one hand, a counter-measure can apply directly on the intrusion objective. This counter-measure, called *kill-login*, kills the login process. On the other hand, we can react before

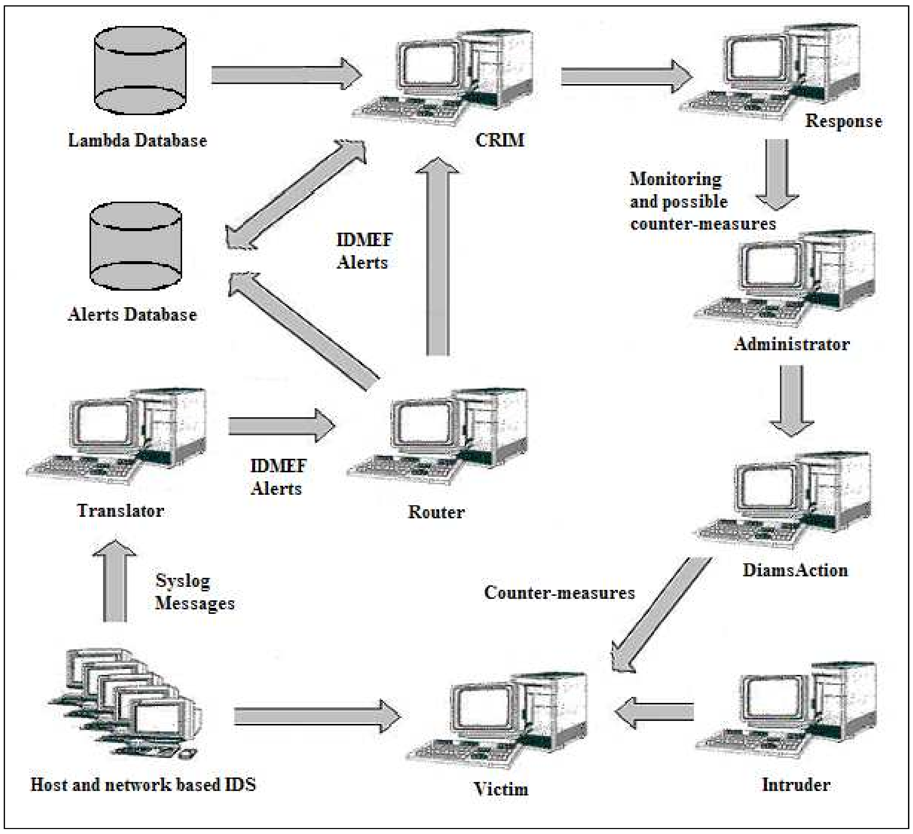

Figure 8: Diams framework

the intrusion objective is achieved. The counter-measure, called *close-connection*, closes the connection[5] between the host and the intruder.

These two counter-measures are presented to the administrator who can select one of them. After a counter-measure is selected, the corresponding script is executed by DIAMS-Action. DIAMS will then send an alert to CRIM to tell that a given counter-measure was launched. Finally, CRIM can apply the *verification* field of the counter-measure to check that that the counter-measure is effective.

7 Conclusion

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 [7]. 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 in DIAMS, a platform of intrusion detection that collects and analyzes alerts generated by various intrusion detection systems. DIAMS is a research prototype implemented in Java whereas the response module is implemented in Prolog. These two modules can interact via JPL (Java Prolog Interface).

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 [11], dynamic changes of the security policy may cause failure of some software components. This is why [11] suggests the notion of security agility, a

[5]by sending a TCP-Reset

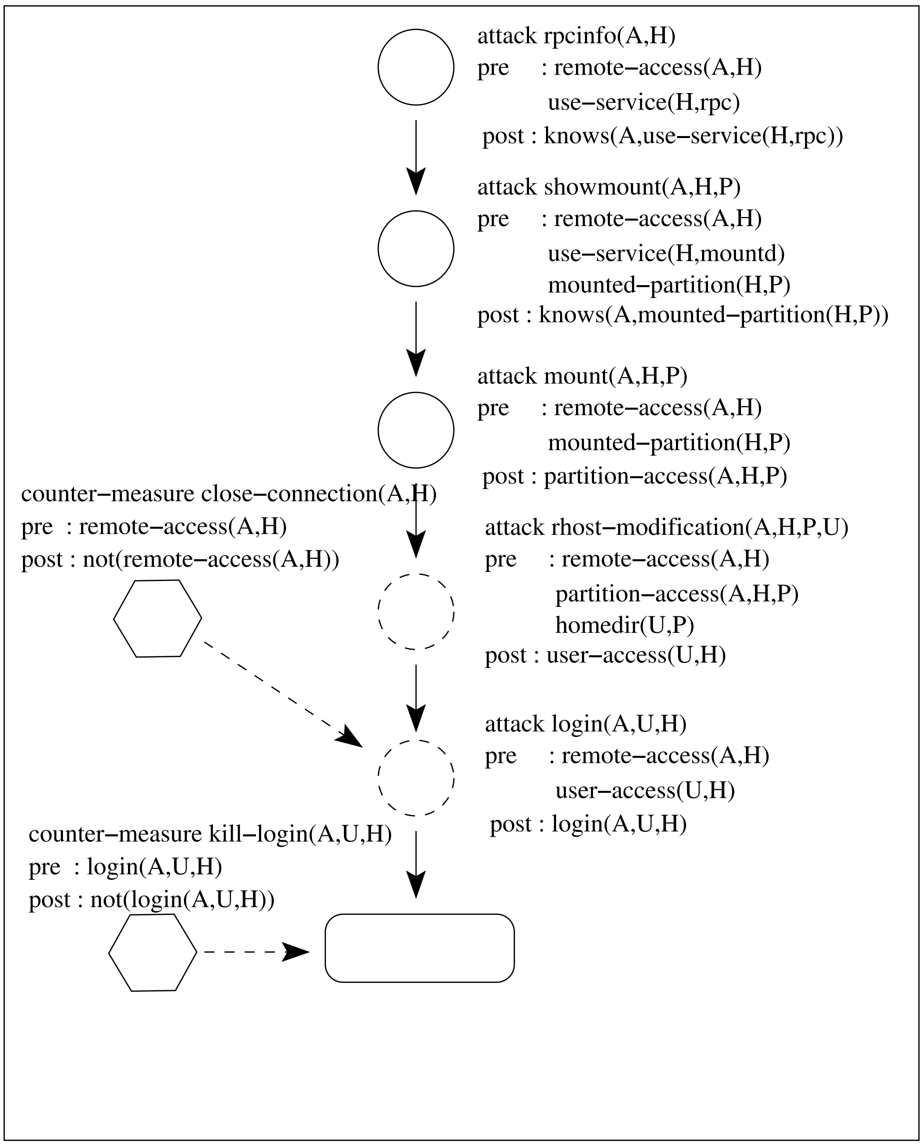

Figure 9: Illegal remote user access

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.

References

[1] R. Bace. *Intrusion Detection.* McMillan, 2000.

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

[3] F. Cuppens, F. Autrel, A. Miège, and S. Benferhat. Recognizing malicious intention in an intrusion detection process. In *Second International Conference on Hybrid Intelligent Systems*, Santiago, Chili, December 2002. Special session: "Hybrid Intelligent Systems for Intrusion Detection".

[4] Frédéric Cuppens. Managing alert in a cooperative intrusion detection environnement. In *17th Annual Computer Security Applications Conference (ACSAC)*, New-Orleans, LA, December 2001.

[5] Frédéric Cuppens and Alexandre Miège. Alert correlation in a cooperative intrusion detection framework. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May 2002.

[6] Frédéric Cuppens and Rodolphe Ortalo. Lambda: A language to model a database for detection of attacks. In H. Debar, L. Mé, and S. F. Wu, editors, *Proceedings of the Third International Workshop on the Recent Advances in Intrusion Detection (RAID'2000)*, number 1907 in LNCS, pages 197–216, October 2000.

[7] Sylvain Gombault and Mamadou Diop. Response function. In *1st Symposium on Real Time Intrusion Detection (NATO)*, Lisbon, Portugal, May 2002.

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

[9] Peng Ning and Dingbang Xu. Learning attack strategies from intrusion alerts. In *10th ACM Conference on Computer and Communication Security*, Washington, DC, 2003.

[10] S. Northcutt and J. Novak. *Network Intrusion Detection*. Paperback, 1999.

[11] Mike Petkac and Lee Badger. Security agility in response to intrusion detection. In *16th Annual Computer Security Applications Conference (ACSAC)*, New-Orleans, LA, 2000.

[12] Martin Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of LISA'99: 13th Systems Administration Conference*, Seattle, Washington, USA, November 7-12 1999.

[13] H.C.M De Swart. *Mathematics, Language, Computer Science and Philosophy*, volume 2. Peter Lang, 1994.