Alert Correlation in a Cooperative Intrusion Detection Framework

Frédéric Cuppens   Alexandre Miège
ONERA Centre de Toulouse
2, av. Edouard Belin
31005, Toulouse CEDEX, France

Abstract

This paper presents the work we have done within the MIRADOR project to design CRIM, a cooperative module for intrusion detection systems (IDS). This module implements functions to manage, cluster, merge and correlate alerts. The clustering and merging functions recognize alerts that correspond to the same occurrence of an attack and create a new alert that merge data contained in these various alerts. Experiments show that these functions significantly reduce the number of alerts. However, we also observe that alerts we obtain are still too elementary to be managed by a security administrator. The purpose of the correlation function is thus to generate global and synthetic alerts. This paper focuses on the approach we suggest to design this function.

Keywords: *Cooperative Intrusion Detection, IDMEF, Alert Merging, Alert Correlation.*

1. Introduction

There are actually two main intrusion detection approaches: the behavioral approach (also called *anomaly detection*) and the signature analysis (also called *misuse detection*). Anomaly detection is based on statistical description of the normal behavior of users or applications. The objective is then to detect any abnormal action performed by these users or applications. The second approach, called misuse detection, is based on collecting attack signatures in order to store them in an attack base. The IDS then parses audit files to find patterns that match the description of an attack stored in the attack base.

None of these approaches is fully satisfactory. They generally generate many false positives (corresponding to a false alert), false negatives (corresponding to a non-detected attack) and the alerts are too elementary and not enough accurate to be directly managed by a security administrator.

For instance, anomaly detection can generate many false positives. This is because deviation from normal behavior does not always correspond to the occurrence of an attack. Moreover, a malicious internal user can *slowly* modify his behavior so that the final behavior includes an attack. The IDS will learn this new behavior and will associate it with a normal behavior. Therefore, the attack will not be detected. This corresponds to the occurrence of a false negative.

The problem of exhaustively defining the attack base is a major difficulty of misuse detection. Therefore, misuse detection can also generate many false negatives especially when a given attack has many close but different implementations. Moreover, in current products, the quality of signatures expressed in the attack base is generally not sufficient to avoid false positives.

In this context, a promising approach is to develop a cooperation module between several IDS to analyze alerts and generate more global and synthetic alerts. The objective is to reduce the number of generated alerts and increase the detection rate of attacks. We also want to provide the security administrator with alerts that can be used to take the right decision.

CRIM is such a cooperative module we have developed within MIRADOR. MIRADOR is a project initiated by the French Defense Agency (DGA) and is led by Alcatel in collaboration with 3 research laboratories: ONERA, ENST-Bretagne and Supelec. MIRADOR aims to build a cooperative and adaptive IDS platform. CRIM is part of this project and implements the following functions: alert clustering, alert merging and alert correlation. A cluster of alerts is a set of alerts that correspond to the same occurrence of an attack. The purpose of the merging function is then to create a new alert that is representative of the information contained in the various alerts belonging to this cluster.

This approach enables us to reduce the number of alerts transmitted to the security administrator. We check our merging function over an attack base of 87 “elementary” attacks. An elementary attack corresponds to a non-decomposable step of a given scenario. For this experiment, we used two different network-based IDS: Snort [12] and e-Trust [1]. The results we obtained were as follows. The 87 attacks generated 325 alerts: 264 for Snort and 61 for e-Trust. Only 69 attacks were detected: 41 by both Snort and e-Trust, 27 by Snort but not by e-Trust, 1 by e-Trust but not by Snort and 18 attacks were not detected. When checking our clustering function on the above attack base, we actually obtained 101 clusters (see [3] for further details on the generation of these clusters).

But, the alerts we obtained still correspond to too elementary alerts. The consequence will be that the security administrator will have difficulty to take the correct decision when receiving these alerts.

Therefore, a complementary analysis must be performed. This is the purpose of the correlation function. The principle of the correlation function is to consider that the intruder wants to achieve a malicious objective but he cannot generally get his way by only performing a single attack. Instead, he usually performs several attacks that correspond to steps of a more global intrusion plan that enables him to achieve his malicious objective. Notice that we include, in the intrusion plan, preliminary steps the intruder generally performs to collect various information on configuration of the system to be attacked.

Classical IDS only detect elementary attacks that correspond to the steps of this intrusion plan. The objective of the correlation function is thus to correlate alerts in order to recognize the intrusion plan that is currently executed by the intruder.

In this paper, we present the approach we suggest implementing the correlation function. The remainder of this paper is organized as follows. Section 1 summarizes the main principles of our approach. We first introduce the architecture of CRIM, the cooperative module we developed for intrusion detection. We shortly presents the objectives of the clustering, merging and correlation functions. We then suggest our approach to modeling alerts and attacks. Both models are based on first order logic and are used in our correlation approach. Actually, our representation of attacks is based on the LAMBDA language [4]. Section 3 sketches our correlation approach, comparing it with other approaches suggested in the literature. Section 4 formalizes this approach and section 5 further refines it by introducing the concept of abductive correlation. Finally, section 6 concludes this paper.

2. General principles

2.1. CRIM Architecture

Figure 1 presents the main principles we suggest to developing a cooperation module for intrusion detection. There are five main functions in this module.

The alert base management function receives the alerts generated by different IDS and stores them for further analysis by the cooperation module. We shall assume that all these alerts are compliant with the Intrusion Detection Message Exchange Format (IDMEF) [2]. The purpose of the IDMEF is to define common data formats and exchange procedures for sharing information of interest to intrusion detection and response systems and those that may need to interact with them.

The approach we suggest to implement the alert base management function is to convert the IDMEF messages into a set of tuples and to store them into a relational database (see section 2.2 below).

The clustering function can then have an access to this database and generates clusters of alerts. When an attack occurs, the IDS connected to CRIM may generate several alerts for this attack. The clustering function attempts to recognize the alerts that actually correspond to the same occurrence of an attack. These alerts are brought into a cluster. As presented in [3], a relation of similarity connects alerts belonging to the same cluster. Each cluster is then sent to the alert merging function. This function was also presented in [3]. For each cluster, this function creates a new alert that is representative of the information contained in the various alerts belonging to this cluster.

The purpose of this paper is to present next step in our

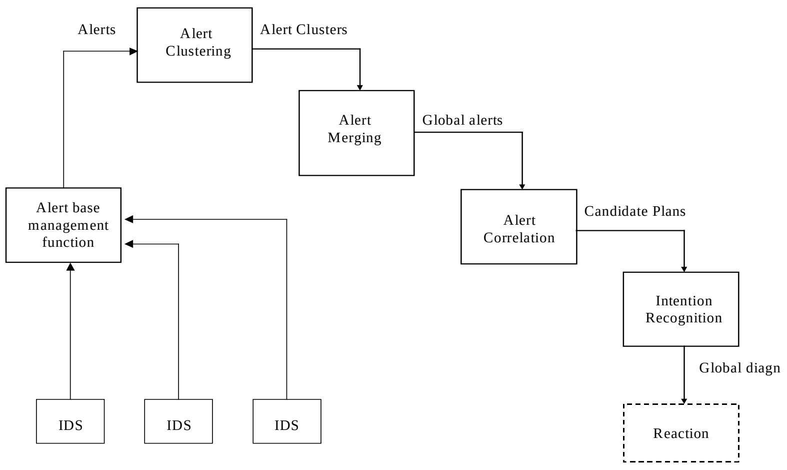


Figure 1: CRIM architecture

cooperation module, that is the correlation function. This function further analyzes the cluster alerts provided as outputs by the merging function. As mentioned in the introduction, we observe that the merging function generally provides too elementary alerts. The objective of the correlation function is thus to correlate alerts in order to provide the security administrator with more synthetic information.

The result of the correlation function is a set of candidate plans that correspond to the intrusion under execution by the intruder. However, the final objective of the intruder is perhaps not achieved yet. Next step in our cooperation module is thus to develop the intention recognition function. The purpose of this function is to extrapolate these candidate plans in order to anticipate the intruder actions. This function should provide a global diagnosis of the past (what the intruder has performed up to now), the present (what the intruder has obtained and what is the current security state of the system targeted by the intruder) and the future (how the intruder will go on). The result of this function is to be used by the reaction function to help the system administrator to choose the best counter measure to be launched to prevent the malicious actions performed by the intruder.

As mentioned in the introduction, we shall only present in this paper our approach to correlating alerts. The intention recognition function is not presented. It is under development and the approach we suggest for this function is briefly sketched in the conclusion of this paper.

2.2. Alert modeling

In our approach, every alert is modeled using the IDMEF format. A Document Type Definition (DTD) has been proposed to describe IDMEF data format through XML documents. This is the representation we shall consider in the remainder of this paper.

However, the correlation function does not directly deal with this XML representation of alerts. It is actually automatically converted into a set of logical facts. This set of facts is then stored in a database.

For instance, let us consider the following portion of an alert in the IDMEF format:

```
<?xml version="1.0"?>
<!DOCTYPE IDMEF-Message PUBLIC "-//IETF//DTD
RFCxxxx IDMEF v0.3//EN" "/usr/sbin/idmef-
message.dtd">
<IDMEF-Message version="0.3">
  <Alert ident="249">
       <Analyzer analyzerid="snort-0000-zp109">
         <Node category="dns">
         <location>unknown</location>
         <name>zp109</name>
       </Node>
         <Process>
           <name>snort</name>
         </Process>
       </Analyzer>
….
</Alert>
</IDMEF-Message>
```

It is translated into the following logical representation (where "," represents conjunction):

```
alert(1),
   ident(1,"249"),
analyzer(1,2),
   analyzerid(2, "snort-0000-zp109"),
   analyzer_node(2,3),
        node_category(3,"dns"),
        node_location(3,"unknown"),
        node_name(3,"zp109"),
        analyzer_process(2,4),
        process_name(4,"snort"),
        ….
```

In this representation, ident, analyzer, analyzerid, etc. are binary predicates we use to represent the alert. Predicates are also introduced to describe other portions of an alert description such as detect time, create time, source, target, classification, etc. Actually, we have defined 34 predicates to completely represent all the possible fields of an alert as suggested in the IDMEF format. Numbers 1, 2, 3, 4, … that appears in the above description correspond to object identifiers that are internally created to represent all the sub-parts of an alert description.

2.3. Attack specification in LAMBDA

Since our approach to alert correlation is based on attacks specified in the LAMBDA language, we shortly recall the main principles of this language (see also [4] for a more detailed presentation).

In this language, an attack is specified using five fields:

- Attack Pre-condition: A logical condition that specifies the conditions to be satisfied for the attack to succeed.
- Attack Post-condition: A logical condition that specifies the effect of the attack when this attack succeeds.
- Attack scenario: The combination of events the intruder performs when executing the attack.
- Detection scenario: The combination of events that are necessary to detect an occurrence of the attack.
- Verification scenario: A combination of events to be launched to check if the attack succeeds.

Notice that other fields might be included in the attack description. For instance, the ADELE language [11] suggests introducing a “reaction” field to specify the actions to be launched when the attack is detected. Actually, in the remainder of this paper, we shall only consider the “Pre-condition”, “Post-condition” and “Detection scenario” fields.

The pre-condition and post-condition of an attack correspond to description of conditions over the *system’s state*. For this purpose, we use a language, denoted L_1, which is based on the logic of predicates. Predicates are used to describe properties of the state relevant to the description of an attack. The set of predicates used to represent these state conditions is partly inspired from the taxonomy suggested by the Darpa to classify attacks (see [8] for a complete presentation of this taxonomy). More specifically, we shall use:

- A predicate to specify the access level of the intruder over the target system: access_level. For example, the fact access_level(bad_guy,192.168.12.3,local) specifies that the user whose name is “bad_guy” has a local access to host 192.168.12.3. Possible values of the access level are remote, local, user, root and physical.
- A set of predicates to specify the effects of attacks on the target system. This set includes predicates deny_of_service, alter and (illegal) use. For instance, the fact deny_of_service(192.168.12.3) specifies that the attack causes a deny of service on host 192.168.12.3.
- Predicates to specify conditions on the state of the source or target systems. For instance use_service(192.168.12.3,showmount) specifies that service showmount is active on host 192.168.12.3.

These predicates are combined using the logical connectives “,” (conjunction denoted by a comma) and “not” (negation). Currently, we do not allow using disjunction in the pre and post descriptions of an attack. Another restriction is that negation only applies to predicates, not to conjunctive expressions.[1]

Figure 2 provides 4 examples of attacks specified in LAMBDA: NFS mount, Modification of .rhost file, TCPScan and Winnuke. In this description, terms starting with an upper case letter correspond to variables and other terms correspond to constants. For instance, pre-condition of NFS mount attack says:

- access_level(Source_user,Target_address,remote),mounted_partition(Target_address,Partition)

that is, to perform NFS mount attack, the intruder Source_user must have a remote access on the target whose IP address is Target_address and Partition must be a mounted partition.

The post condition of this attack says:

- can_access(Source_user,Partition)

That is the intruder Source_user gets an access on the mounted partition Partition.

Notice that sometimes the effect of an attack is simply a knowledge gain for the attacker about the target system. This is for instance the case of attack TCPScan in figure 2. Describing this kind of attacks is very important since their occurrence often corresponds to preliminary steps of a more global attack scenario. In order to represent a knowledge gain, we extend language L_1 so that it also includes a meta-predicate (actually a logical modality) *knows*. For instance, if bad_guy is the attacker, then knows(bad_guy, use_service(192.168.12.3,’NetBios’)) means that bad_guy knows that *NetBios* is an active service of system whose IP address is 192.168.12.3.

The other fields of an attack description in LAMBDA correspond to attack scenario, detection scenario and verification scenario[2]. These scenarios are specified using event calculus algebra. This algebra enables us to combine several events using operators such as: ; (sequential composition), | (parallel unconstrained execution), ? (non deterministic choice), & (synchronized execution) and if_not (exclusion of an event when another event occurs). However, all the examples of attacks we shall use in this paper (including examples presented in figure 2) actually correspond to elementary scenarios based on a single event. This is represented by:

- <scenario>Action</scenario>

  to specify that Action is the single event corresponding to the attack scenario.
- <detection>Alert</detection>

  to specify that Alert is the single event corresponding to the detection of the attack.

Finally, conditions appearing in fields cond_scenario and cond_detection are used to formulate description of the event specified in the scenario and detection fields. The cond_scenario field is generally specified using the *script* predicate to represent the command the intruder runs to perform the attack. The cond_detection field is used to describe the main attributes of the alert we expect when the attack occurs. This corresponds to a logical expression without restriction (that is, it can include conjunction, disjunction or negation). It is built using the predicates we introduced in section 2.2 to logically model an alert. For instance, expression:

- alert(Alert), classification(Alert,"MIR-0163"),
  source(Alert,Source),
  source_user(Source,Source_user)

[1] The reason of these restrictions will be explained in section 4.2.

[2] Actually, description of the verification_scenario field is not provided in the examples of figure 2.

```
<?xml version="1.0" encoding="UTF-8"?>
<attack attackid="MIR-0163">
<name>mount partition</name>
<pre>access_level(Source_user,Target_address,remote),
        mounted_partition(Target_address,Partition),
</pre>
<post>can_access(Source_user,Partition)
</post>
<scenario>Action</scenario>
<cond_scenario>
  script(Action,’mount -t nfs $Partition:$Target_address $Partition’)
</cond_scenario>
<detection>Alert</detection>
<cond_detection>alert(Alert),
                       source(Alert,Source),
                       source_user(Source,Source_user),
                       target(Alert,Target),
                       target_node(Target,Target_node),
                       address(Node,Target_address),
                       classification(Alert,"MIR-0163")
</cond_detection>
</attack>
```

Lambda attack MIR-0163 – NFSMount

```
<?xml version="1.0" encoding="UTF-8"?>
<attack attackid="MIR-0164">
<name>modification du .rhost</name>
<pre>access_level(Source_user,Target_address,remote),
       can_access(Source_user,Partition),
       owner(Partition,Target_User),
       userid(Target_user,Target_address,Userid),
</pre>
<post> access_level(Source_user,Target_address,user)
</post>
<scenario>Action</scenario>
<cond_scenario>script(Action,’cat "++" > .rhost’)</cond_scenario>
<detection>Alert</detection>
<cond_detection>alert(Alert),
                       source(Alert,Source),
                       source_user(Source,Source_user),
                       target(Alert,Target),
                       target_node(Target,Target_node),
                       address(Target_node,Target_address),
                       classification(Alert,"MIR-0164")
</cond_detection>
</attack>
```

Lambda attack MIR-0163 – Modification of .rhost

```
<?xml version="1.0" encoding="UTF-8"?>
<attack attackid="MIR-0036">
<name>winnuke sur la cible</name>
<pre>use_os(Target_address,windows),
              state(Target_address,available),
              dns_server(Target_address)
</pre>
<post>deny_of_service(Target_address)
</post>
<scenario>Action</scenario>
<cond_scenario>
   script(Action,’winnuke $Target_address’)
</cond_scenario>
<detection>Alert</detection>
<cond_detection>alert(Alert),
                            source(Alert,Source),
                            source_node(Source,Source_node),
                            address(Source_node,Source_address),
                            target(Alert,Target),
                       target_node(Target,Target_node),
                       address(Target_node,Target_address),
                       classification(Alert,"MIR-0036")
</cond_detection>
</attack>
```

Lambda attack MIR-0036 – Winnuke

```
<?xml version="1.0" encoding="UTF-8"?>
<attack attackid="MIR-0074">
<name>tcpscan sur la cible</name>
<pre>use_soft(Source_address,tcpscan),
        use_service(Target_address,Target_service),
       service_type(Target_service,tcp)
</pre>
<post>
  knows(Source_user,use_service(Target_address,Target_service))
</post>
<scenario>Action</scenario>
<cond_scenario>script(Action,’tcpscan $Target_address’)
</cond_scenario>
<detection>Alert</detection>
<cond_detection>alert(Alert),
                       source(Alert,Source),
                       source_node(Source,Source_node),
                       address(Source_node,Source_address),
                       source_user(Source,Source_user),
                       target(Alert,Target),
                       target_node(Target,Target_node),
                       target_service(Target,Target_service),
                       classification(Alert,"MIR-0074")
</cond_detection>
</attack>
```

Lambda attack MIR-0074 – TCPScan

Figure 2 : Attack specification in Lambda

specifies that Alert is analert whose classification is "MIR-0163" and source must match a given variable Source. The user associated with this source is another variable Source_user. This description enables us to formulate constraints between the various fields of an alert and the variables used in the pre_condition and post_condition description of an attack.

Notice that, in the following, the alert classification will have always the form “MIR-xxxx”. This corresponds to an internal classification of attacks used within the MIRADOR project. This classification is used by the merging function to translate “vendor specific” classification into a common classification so that it is thus possible to make correspondence between alert classifications generated by two different IDS (see [3] for more details).

3. Explicit correlation

We identified two main approaches to achieve correlation[3]:

- Explicit correlation of events is used when the security administrator is able to express some connection between events that he knows. This connection may be a logical link based on knowledge of relations between alerts. It may be also a link depending on the topology of information system's components [7,6].
- Implicit correlation of events is used when data analysis brings out some mappings (may be statistical ones) and relations between events. This approach is mainly based on observing groups of alerts and extracting implicit relations between them. Many works show that intrusion detection probes produce groups of alerts according to the configuration data, the traffic and the topology of information system under surveillance. Such approaches are based on learning techniques (classification [9], data mining [15], neural network [10], …) and should significantly reduce the amount of alerts we have to deal with.

We opt for the explicit approach to carry out the correlation function. Thus, it must be possible to express explicitly known logical links between attacks. This is achieved by using the following predicate:

- attack_correlation(Attack1,Attack2): this predicate says that Attack1 may be correlated with Attack2, that is Attack1 enables the intruder to then perform Attack2.

[3] Notice that several authors use the terms “alert correlation” for functions that actually correspond to “alert merging” in our terminology (see [13] for instance). We do not consider such approaches in the remainder of this paper.

For instance, the fact attack_correlation(“MIR-0066”,”MIR-0162”) specifies that it is possible to correlate attack “MIR-0066” (which corresponds to “rpcinfo”) with attack “MIR-0162” (which corresponds to “showmount”). This is because the attack “rpcinfo” enables the intruder to learn if rpc service “showmount” is active.

However, our objective is not to correlate attacks but to correlate alerts. Predicate “attack_correlation” is too coarse for this purpose because it does not provide conditions the alerts must satisfy to correlate them. For instance, we can only consider that alerts corresponding to attacks “MIR-0066” and “MIR-0162” are steps of the same scenario if the target systems appearing in these alerts are equal. If this condition is not satisfied, there is no reason to correlate these alerts.

Therefore, we also introduce the following predicate:

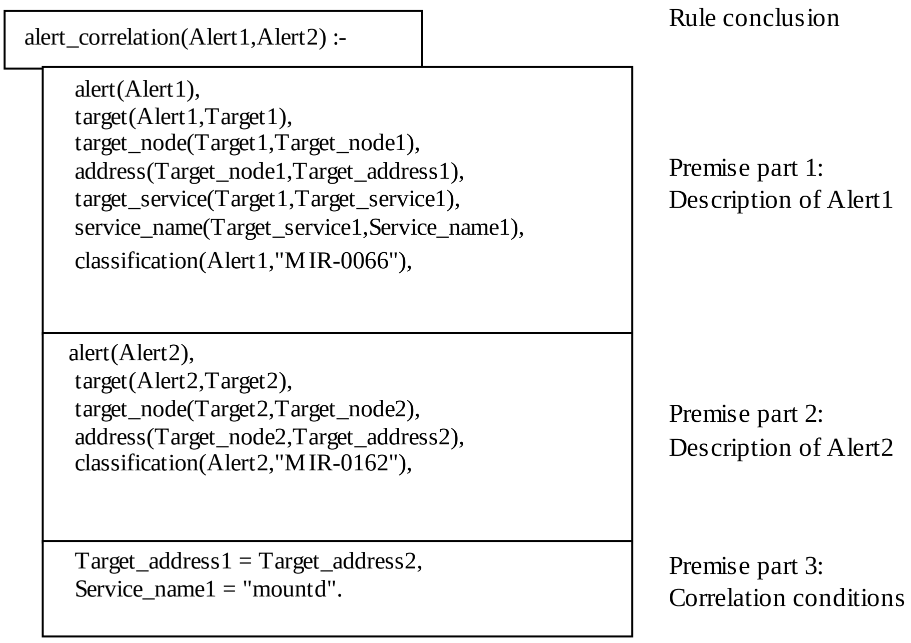
- alert_correlation(Alert1,Alert2): this predicate says that Alert1 is correlated with Alert2.

This predicate is used to specify *correlation rules*. The conclusion of a correlation rule has always the form alert_correlation(Alert1,Alert2). Its premise specifies the conditions Alert1 and Alert2 must satisfy to conclude that Alert1 and Alert2 can be correlated as part of a given attack scenario.

For instance, figure 3 presents the rule to correlate two alerts whose classifications are respectively “MIR-0066” and “MIR-0162”. The premise of a correlation rule has always three parts. Part 1 and 2 respectively provide a description of the two alerts to be correlated. These descriptions correspond to logical conditions expressed in the language presented in section 2.2 to model alerts. Part 3 of the premise expresses the conditions to be satisfied to correlate the two alerts. In the above example, there are two such conditions:

- Condition 1 says that the target addresses appearing in the alerts must be equal (meaning that attacks rpcinfo and showmount applies to the same target)
- Condition 2 says that the service name appearing in Alert1 must be equal to service “mountd” (meaning that one of the services provided by rpcinfo is equal to “mountd”)

Notice that there is always an implicit condition to correlate two alerts Alert1 and Alert2. This condition says that Alert1 must occur before Alert2. It is checked by comparing the detect time of Alert1 with the detect time of Alert2. However, this condition is not directly expressed in the correlation rules because it would simply burden specification. Instead, it is systematically checked when the correlation rules are evaluated (see section 4.5 for more details on application of correlation rules).



```
alert_correlation(Alert1,Alert2) :-
    alert(Alert1),
    target(Alert1,Target1),
    target_node(Target1,Target_node1),
    address(Target_node1,Target_address1),
    target_service(Target1,Target_service1),
    service_name(Target_service1,Service_name1),
    classification(Alert1,"MIR-0066"),

    alert(Alert2),
    target(Alert2,Target2),
    target_node(Target2,Target_node2),
    address(Target_node2,Target_address2),
    classification(Alert2,"MIR-0162"),

    Target_address1 = Target_address2,
    Service_name1 = "mountd".
```

Figure 3: Example of correlation rule between alerts corresponding to attacks "MIR-0066" (rpcinfo) and "MIR-162" (showmount)

As the reader may notice, specifying correlation rules would be a quite complex task to perform manually:

- It would be tedious for the administrator at least from a syntactical point of view.
- It is not obvious to be exhaustive, that is not to forget correlation rules specifying pairs of alerts that may be correlated.
- It is also not always obvious to specify the right correlation conditions.

This is the reason why it would be very interesting to have a method to automatically generate correlation rules. We have developed such a method, called semi-explicit correlation. It is presented in the following section.

4. Semi-explicit correlation in LAMBDA

4.1. Background of the approach

This section presents the approach we suggest to performing alert correlation. This approach is based on the analysis of attack description specified in LAMBDA. The central idea of the approach is to recognize whether executing a given attack can *contribute* to execute another attack.

This idea is modeled by specifying possible logical links between the post-condition of an attack A and the pre-condition of an attack B. If such a link exists, then it is possible to correlate an occurrence of attack A with an occurrence of attack B because we can assume that the intruder has performed A as a step that enables him to perform B.

To formally define this kind of correlation between the post-condition of an attack and the pre-condition of another attack, let post(A) be the logical formula representing the post-condition of attack A and pre(B) be the logical formula representing the pre-condition of attack B. Of course, we can correlate attack A and attack B if post(A) logically implies pre(B), that is:

$$\text{post}(A) \rightarrow \text{pre}(B)$$

However, this definition is generally too strong. This is because it is sufficient to correlate attack A with attack B that attack A "contributes" to the realization of attack B. This is formally specified as follows:

$$\text{post}(A) \wedge \text{hyp} \rightarrow \text{pre}(B)$$

where hyp is an hypothesis that, when combined with post(A), implies pre(B). Of course, the hypothesis hyp alone must not be sufficient to imply pre(B), that is we must not have hyp $\rightarrow$ pre(B). Another requirement is that hyp must be consistent with post(A) because if this is not the case then one can derive anything from post(A) $\wedge$ hyp, in particular pre(B).

The work we have done is based on this general definition of correlation. However, this first definition is not very manageable. Next sections present more practical definitions of correlation. The implementation of the correlation function in CRIM is actually based on these definitions.

4.2. Definition of alert correlation

Let A and B be two attacks and let Post(A) and Pre(B) respectively be the post condition of attack A and pre condition of attack B. Let us assume that Post(A) and Pre(B) respectively have the following form:[4]

- Post(A) = $expr_{A1}$, $expr_{A2}$, … , $expr_{Am}$
- Pre(B) = $expr_{B1}$, $expr_{B2}$, … , $expr_{Bn}$

where each $expr_i$ must have one of the following forms:

- $expr_i$ = pred
- $expr_i$ = not(pred)
- $expr_i$ = knows(User,pred)
- $expr_i$ = knows(User, not(pred))

where pred is a predicate.

Definition 1: Direct correlation (simple case)

We say that attack A and attack B are directly correlated if the following condition is satisfied:

- there exists i in [1,m] and j in [1,n] such that $expr_{Ai}$ and $expr_{Bj}$ are unifiable through a most general unifier (mgu) θ.

For instance, attacks "MIR-0163" (NFS Mount) and "MIR-0164" (Modification of .rhost) are directly correlated. This is because post("MIR-0163") is equal to can_access(Source_user,Partition) and this predicate also appears in pre("MIR-0164"). After renaming the variables of can_access(Source_user,Partition) that respectively appear in post("MIR-0163") and pre("MIR-0164") into can_access(Source_user1,Partition1) and can_access(Source_user2,Partition2), we can conclude that these expressions are unifiable through mgu θ such that Source_user1 = Source_user2 and Partition1 = Partition2.

On the other hand, the converse is not true, that is attack "MIR-0164" is not directly correlated with attack "MIR-0163". This is because post("MIR-0164") is equal to access_level(Source_user,Target_address,user). Predicate access_level(Source_user,Target_address,remote) appears in pre("MIR-0163") but since constants *user* and *remote* are not unifiable, correlation of "MIR-0164" with "MIR-0163" fails.

Let us now try to correlate attack "MIR-0162" (Showmount) with attack "MIR-0163" (Mount partition). A possible post condition of "MIR-0162" is knows(Source_user,mounted_partition(Target_address, Partition)), that is the intruder Source_user knows what partitions are mounted on the target whose IP address is Target_address. On the other hand, mounted_partition(Target_address,Partition) appears in pre("MIR-0163"). However, due to *knows* modality, this last expression is not directly unifiable with post("MIR-0162"). This is intuitively not satisfactory since executing Showmount enables the intruder to then mount a partition observed in Showmount.[5]

Therefore, we slightly modify definition 1 so that attack "MIR-0162" can be correlated with "MIR-0163". This leads to the following definition:

Definition 2: Direct correlation (general case)

We say that attack A and attack B are directly correlated if one of the following conditions is satisfied:

- there exists i in [1,m] and j in [1,n] such that $expr_{Ai}$ and $expr_{Bj}$ are unifiable through a mgu θ.

  or
- there exists i in [1,m] and j in [1,n] such that $expr_{Ai}$ and knows(User,$expr_{Bj}$) are unifiable through a mgu θ.

4.3. Indirect correlation

Let us now consider attacks "MIR-0073" (TCPScan) and "MIR-0036" (Winnuke"). These two attacks are not correlated using definition 2. However, attack Winnuke to succeed requires that the operating system used on the target system is Windows. The intruder can obtain this knowledge about the target system by performing TCPScan and by observing that port 139 is open (meaning that a NetBios session is open which is characteristic of Windows system).

Therefore, it would be suitable to correlate attacks "TCPScan" and "Winnuke" in the case where port 139 is scanned (and open). For this purpose, the solution we suggest is to specify ontological rules to represent possible relations between predicates. These ontological rules are also represented using a pre and post condition.

Figure 4 shows an example of such a rule. This ontological rule says that if a system whose IP address is System_address uses service NetBios, then the operating system used on this system is Windows.

[4] Notice that we assume that the pre and post conditions do not include disjunction. This is a restriction that is used to simplify definition of correlation below. From a practical point of view, including disjunction in the pre condition does not really increase the expressive power of our attack description language since disjunctions in the pre condition might be split into several sub-rules corresponding to each part of the disjunction. On the other hand, disjunction in the post condition is useful since it would enable us to specify some non-determinism in the effect of an attack. So, generalyzing correlation definitions below to take into account such disjunctions represents further work that remains to be done.

[5] To justify this point we actually assume that modality *knows* satisfies the following axion for each *User* and *Expr*: *knows(User,Expr)* → *Expr*, that is if *User* knows that *Expr* then *Expr* is true.

```
<?xml version="1.0" encoding="UTF-8">
<rule ruleid="RULE-0001">
   <pre>
      use_service(System_address,'NetBios')
   </pre>
   <post>
      use_os(System_address,windows)
   </post>
</rule>
```

Figure 4: Example of ontological rule

From a syntactical point of view, we assume that restrictions that apply to the representation of pre and post conditions in an ontological rule are similar to the one for pre and post conditions of an attack (that is, they do not include disjunction).

Next step is then to generalize definition 2 when ontological rule are used to perform correlation. This generalization is done in two steps. We first generalize definition 2 so that it applies to correlate two ontological rules or an attack with an ontological rule or an ontological rule with an attack. Since we assume that the syntactical format of the pre and post conditions of an attack is similar to the one of an ontological rule, this generalization is straightforward.

We then introduce the notion of indirect correlation. It is defined as follows:

Definition 3: Indirect correlation

We say that attack A and attack B are indirectly correlated through ontological rules R_1, …, R_n if the following conditions are satisfied:

- Attack A is directly correlated with rule R_1 through a most general unifier θ_0,
- For each j in [1,n-1], rule R_j is directly correlated with rule R_{j+1} through a most general unifier θ_j,
- Rule Rn is directly correlated with attack B through a most general unifier θ_n.

Using definition 3, we can now conclude that attack "MIR-0073" (TCPScan) is indirectly correlated with attack "MIR-0036" (Winnuke). This is because the post-condition of attack "MIR-0073" is equal to knows(Source_user,use_service(Target_address,Target_service)). Then, since the pre-condition of "RULE-0001" is equal to use_service(System_address,'NetBios'), "MIR-0073" is directly correlated to "RULE-0001" through the mgu:

- Target_address = System_address, Target_service = 'NetBios'

Similarly, the post-condition of "RULE-0001" is equal to use_os(System_address,windows). Since this predicate also appears in the precondition of "MIR-0036", "RULE-0001" is correlated with "MIR-0036" when System_address = Target_address. Thus, we can conclude that attack "MIR-0073" is indirectly correlated with "MIR-0036".

4.4. Generating correlation rules

In this section, we show how to automatically generate correlation rules similar to the one presented in section 3. The process we suggest is the following.

Let us consider two attacks Attack1 and Attack2 whose descriptions are correlated according to definition 2 through a mgu θ. After renaming the variables that appear in the descriptions of Attack1 and Attack2 so that there is no common variable in these descriptions, we shall generate a correlation rule having the following form:

```
correlation_rule(Alert1,Alert2) :-
    cond_detection(Attack1),
    cond_detection(Attack2),
    θ.
```

where Alert1 and Alert2 are respectively the (renamed) variables that appear in the detection field of Attack1 and Attack2[6]. For example, figure 5.a presents the correlation rule corresponding to attacks "MIR-0163" (NFS Mount) and "MIR-0164" (Modification of .rhost).

This rule is correct but it is not fully optimized[7]. In particular, target descriptions of the two alerts might be removed since they are not related to the correlation condition. Notice also that our process also generates condition Partition1 = Partition2. This is correct since, in this attack scenario, the intruder must modify the .rhost file of a partition previously mounted with attack NFS Mount. But, as Partition1 and Partition2 remains free variables, this condition will be always evaluated to true. This is because we assume that information about the mounted partition is not provided by alerts corresponding to NFS Mount and Modification of .rhost.

The case where two attacks Attack1 and Attack2 are indirectly correlated using ontological rules is slightly more complicated. If Attack1 and Attack2 are indirectly correlated using ontological rules R_1, …, R_n through a set of mgu θ_0, …, θ_n, then we shall generate a correlation rule having the following form:

[6] Notice that our approach does not apply to the case where the detection field of attacks corresponds to combined events. It is restricted to detection scenarios that are single events.

[7] Defining an algorithm to optimize correlation rules might be done. But, the overhead due to this lack of optimization is marginal so that we do not find that such an optimization is a priority.

```
correlation_rule(Alert1,Alert2) :-
    cond_detection(Attack1),
    cond_detection(Attack2),
    θ0, …, θn.
```

For example, figure 5.b presents the correlation rule corresponding to attacks “MIR-0073” (TCPScan) and “MIR-0036” (Winnuke).

Notice that all the correlation rules are automatically generated by analyzing the descriptions in LAMBDA of the set of attacks. This process is performed offline and therefore, it is not time consuming for online intrusion detection.

4.5. Applying correlation rules

After all the correlation rules are generated offline, the online correlation process can start. When this process receives a new alert Alert1, it proceeds as follows.

Let Attack1 be the classification associated with Alert1. In a first step, we check if there are other alerts already stored in the database and whose classification is Attack2 such that the fact attack_correlation(Attack1,Attack2) or attack_correlation(Attack2,Attack1) is stored in the correlation base. Notice that this first step is only for optimization since the correlation rules might be applied directly. However it is more efficient to first filter on predicate attack_correlation to check if there are alerts that are potentially correlated to Alert1. Notice that we both look for facts attack_correlation(Attack1,Attack2) and attack_correlation(Attack2,Attack1) because we do not assume that the alerts are received in the same order as their order of occurrence.

If there are alerts Alert2 that are potentially correlated with Alert1, then the corresponding correlation rules apply to check if the correlation conditions are satisfied. The result is a set of pairs of alerts that are correlated, one member of these pairs being Alert1.

For each pair in this set, we then apply an algorithm to check if this pair might be aggregated to an existing scenario. Else, a new scenario starting with this pair of alerts is generated. For instance, let us assume that there is already a scenario with three alerts (alert1, alert2, alert3). Let us assume that alert4 is received and that the online correlation process generates a pair (alert3, alert4). In this case, a “longer” scenario (alert1, alert2, alert3, alert4) is generated.

Notice that a complex scenario with several branches is actually decomposed into several sequential scenarios corresponding to each branch. For instance, let us consider the scenario presented in figure 6. It is represented by two sequential scenarios (alert1, alert2, alert3, alert4) and (alert2,alert5,alert6,alert4). It will be the role of the graphic interface to “aggregate” these two sequential scenarios as presented in figure 6 (see annex 1 for a short presentation of this interface).

For each sequential scenario, the online correlation process generates a special alert called “scenario alert”. This alert is fully compliant with the IDMEF format.

```
alert_correlation(Alert1,Alert2) :-

  alert(Alert1),
  source(Alert1,Source1),
  source_user(Source1,Source_user1),
  target(Alert1,Target1),
  target_node(Target1,Target_node1),
  address(Target_node1,Target_address1),
  classification(Alert1,"MIR-0163"),

  alert(Alert2),
  source(Alert2,Source2),
  source_user(Source2,Source_user2),
  target(Alert2,Target2),
  target_node(Target2,Target_node2),
  address(Target_node2,Target_address2),
  target_user(Target2,Target_user2),
  classification(Alert2,"MIR-0164"),

  Source_user1 = Source_user2,
  Partition1 = Partition2.
```

Figure 5.a: Correlation rule for “MIR-0163” NFSMount) and “MIR-0164” (Modification of .rhost)

```
alert_correlation(Alert1,Alert2) :-

  alert(Alert1),
  source(Alert1,Source1),
  source_node(Source1,Source_node1),
  address(Source_node1,Source_address1),
  source_user(Source1,Source_user1),
  target(Alert1,Target1),
  target_node(Target1,Target_node1),
  address(Target_node1,Target_address1),
  target_service(Target1,Target_service1),
  classification(Alert1,"MIR-0073”),

  alert(Alert2),
  source(Alert2,Source2),
  source_node(Source2,Source_node2),
  address(Source_node2,Source_address2),
  target(Alert2,Target2),
  target_node(Target2,Target_node2),
  address(Target_node2,Target_address2),
  classification(Alert2,"MIR-0036"),

  Target_address1 = System_address3,
  Target_service1 = ‘NetBios’,

  System_address3 = Target_address2.
```

Figure 5.b: Correlation rule for “MIR-0073” (TCPScan) and “MIR-0036” (Winnuke)

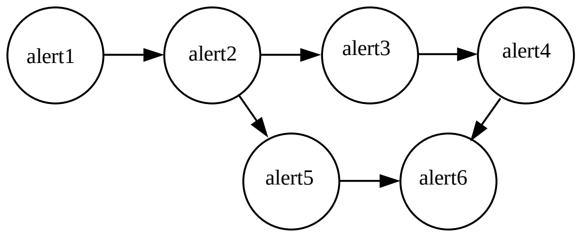


Figure 6: Example of attack scenario

The “Correlationalert” field of this alert corresponds to the list of correlated alerts (the order in this list is important!). The other fields of this alert are generated by using the merging function to merge data contained in the correlated alerts (see [3] for more details about the merging function).

5. Abductive correlation

There are still some problems that arise when we apply our online correlation process. For instance, let us consider the attack scenario presented in figure 7. We call this attack “illegal nfs mount”. This attack scenario enables the intruder to get a root access by exploiting a misconfiguration in the security policy, namely the intruder can mount a partition corresponding to the home directory of root. There are 6 different steps in this attack.

Intrusion scenario	**Detection results**	**Fusion results**
Step 1 : attack_finger **finger** root	Snort : **3** alerts eTrust : 0 alert	CRIM : **1** alert
Step 2 : attack_rpcinfo **rpcinfo** <target>	Snort : **1** alert eTrust : **1** alert	CRIM : **1** alert
Step 3 : attack_showmount **showmount** <target>	Snort : **1** alert eTrust : 0 alert	CRIM : **1** alert
Step 4 : attack_mount **mount** directory	Snort : **1** alert eTrust : 0 alert	CRIM : **1** alert
Step 5 : attack_rhost cat “++” > **.rhost**	**Not detected**	
Step 6 : attack_rlogin **rlogin** <target>	Snort : **1** alert eTrust : **1** alert	CRIM : **1** alert

Figure 7: “Illegal NFS Mount” scenario

These 6 steps are specified in Lambda so that when we apply the offline correlation process, we obtain 6 correlation rules as shown in Figure 8. cond1, … cond6 correspond to the 6 correlation conditions associated with these correlation rules.

The result provided by the offline correlation process should enable the online correlation process to fully recognize the “illegal NFS mount” scenario. However, when this attack is launched on a system supervised by Snort and e-Trust, 9 alerts are generated: 7 by Snort and 2 by e-Trust. Our clustering function gives 5 clusters. Actually, both Snort and e-Trust did not detect step 5.

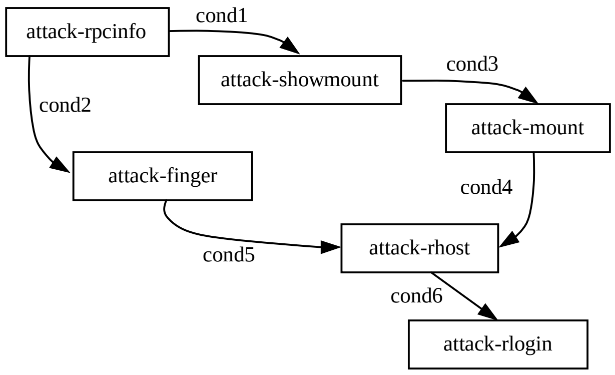


Figure 8: results of correlation process on attack “illegal NFS mount”

This result is then provided to the correlation function for further analysis, the objective being to correlate these 5 clusters in order to recognize one single complex attack. The main difficulty to fully recognize multiple steps attack “illegal NFS mount” comes from the fact that step 5 (“MIR-0164”: attack_rhost) is not detected by both Snort and e-Trust. Our approach to solve this problem is the following.

The correlation function receives one alert corresponding to attack “MIR-0163” (attack_mount) and another one corresponding to “MIR-0165” (attack_rlogin). Since the correlation function knows that it is possible to correlate attack_mount with attack_rhost and then attack_rhost with attack_rlogin, the correlation function makes the hypothesis that it is possible to *transitively* correlate attack_mount with attack_rlogin[8]. In this case, the approach is to abduce[9] a new alert that is to create a new (virtual) alert corresponding to attack_rhost.

When this virtual alert is generated, its classification field is initialized to “MIR-0164” (corresponding to attack_rhost). All the other fields are initialized by using Skolem constants[10]. For instance, the target field is initialized to “target(1)” meaning that the target of this alert exists but is currently unknown. The case of detect time field is slightly more complex. It is partly unknown but must satisfy the following constraint: it must be after the alert corresponding to attack_mount and before the alert corresponding to attack_rlogin. The solution in this

[8] Similar reasoning applies to transitively correlate attack_finger with attack_rlogin.

[9] Abduction consists is making new hypotheses and to use them to derive new facts. Typically, this kind of reasoning is used when facts are missing to complete a diagnostic.

[10] Skolemization is a process used to replace expressions having the form $\exists x, p(x)$ by $p(\alpha)$ where α is a Skolem constant.

case is to manage interval of time. Due to space limitation, this point is not developed here but we plan to present it in a forthcoming paper.

Once the virtual alert is abduced, the online correlation process applies the corresponding correlation rules. It first checks if cond4 is satisfied. In particular, we have to satisfy the following condition: Target_address1 = Target_address2 where Target_address1 is the IP address of alert corresponding to attack_mount and Target_address2 is the IP address of the virtual alert. Since Target_address2 is actually a Skolem constant, this unification succeeds and Target_address2 is updated so that it is now equal to Target_address1. Then, we have to check if cond6 is satisfied. cond6 includes a condition having the form Target_address2 = Target_address3 where Target_address2 is the IP address of the virtual alert and Target_address3 is the IP address of alert corresponding to attack_rlogin. But Target_address2 is now equal to Target_address1, so that checking cond6 will only succeed if Target_address1 is actually equal to Target_address3, that is the alerts corresponding to attack_mount and attack_rlogin have a target with the same IP address. Else cond6 is not satisfied and therefore the abductive correlation does not succeed.

We shall proceed similarly for cond5 to check whether it is possible to correlate the alert corresponding to attack_finger with the virtual alert. In this case, we have also to satisfy the following condition: Target_address4 = Target_address2 where Target_address4 is the IP address of alert corresponding to attack_finger and Target_address2 is the IP address of the virtual alert. This condition is satisfied if the alerts corresponding to attack_mount and attack_finger have a target with the same IP address.

In our experiment, abduction of an alert corresponding to attack_rhost succeeds. Therefore, the online correlation function correlates all the alerts generated by the attack “illegal nfs mount” into one single scenario, even though step 5 of this attack is not detected (see the appendix for a graphical presentation of the corresponding detection).

6. Conclusion

In this paper, we have presented the approach we suggest designing the correlation function of CRIM, a cooperative module for intrusion detection systems. After specifying an attack base in Lambda, the offline correlation process analyzes these attack descriptions to automatically generate a set of correlation rules. The online correlation process then applies these correlation rules on the alerts generated by the IDS to recognize more global attack scenarios.

All the approach (including clustering, merging, correlation and abductive correlation) has been implemented in GNU-Prolog [5] with a graphic interface in Java (see the appendix for a view of this graphical interface).

It is important to observe that alert correlation is very useful to reduce the number of false positives. For instance, notice that separately each step of the intrusion scenario “illegal NFS Mount” presented in section 5 might actually correspond to a false positive. It is only after correlating alerts that we can derive that an intrusion occurred. Therefore, in many cases, it is possible to conclude that an alert that is not further correlated with other alerts is actually a false positive.

There are several issues to this work.

First, we are currently designing the intention recognition function. The main objective of this function is to anticipate on how the intruder will go on. To achieve this objective, we are actually combining two approaches:

- Abductive correlation is used to forecast next step of the attack scenario. This first approach is based on the analysis of facts attack_correlation(Attack1,Attack2) and is therefore simpler than the abductive correlation process presented in section 5 since virtual alerts are not generated in this case.
- Specification of global intrusion objectives. This idea is quite similar to specifying a security policy since we may assume that the intruder’s objective is to violate the security policy (at least from a defensive point of view!). A global intrusion objective might be viewed as a logical expression describing the target’s state the intruder wants to achieve. The principal of this second approach is then to correlate attacks with intrusion objective (corresponding to the violation of the security policy). This would guide the intention recognition process.

Of course, in both cases, the intention recognition function may provide several possibilities, that is several “next steps” in the first approach and several possible intrusion objectives in the second approach. Therefore, we have also to design an approach to choose the “best” possibility. This is still an open research problem.

This briefly sketch our approach for the intention recognition function. We plan to provide more details about this function in a forthcoming paper.

Second, we plan to encode a larger base of attacks in LAMBDA. The objective here is to improve the correlation results but also to check whether it is possible to use our correlation approach to discover new attack scenarios by searching how to correlate elementary attacks.

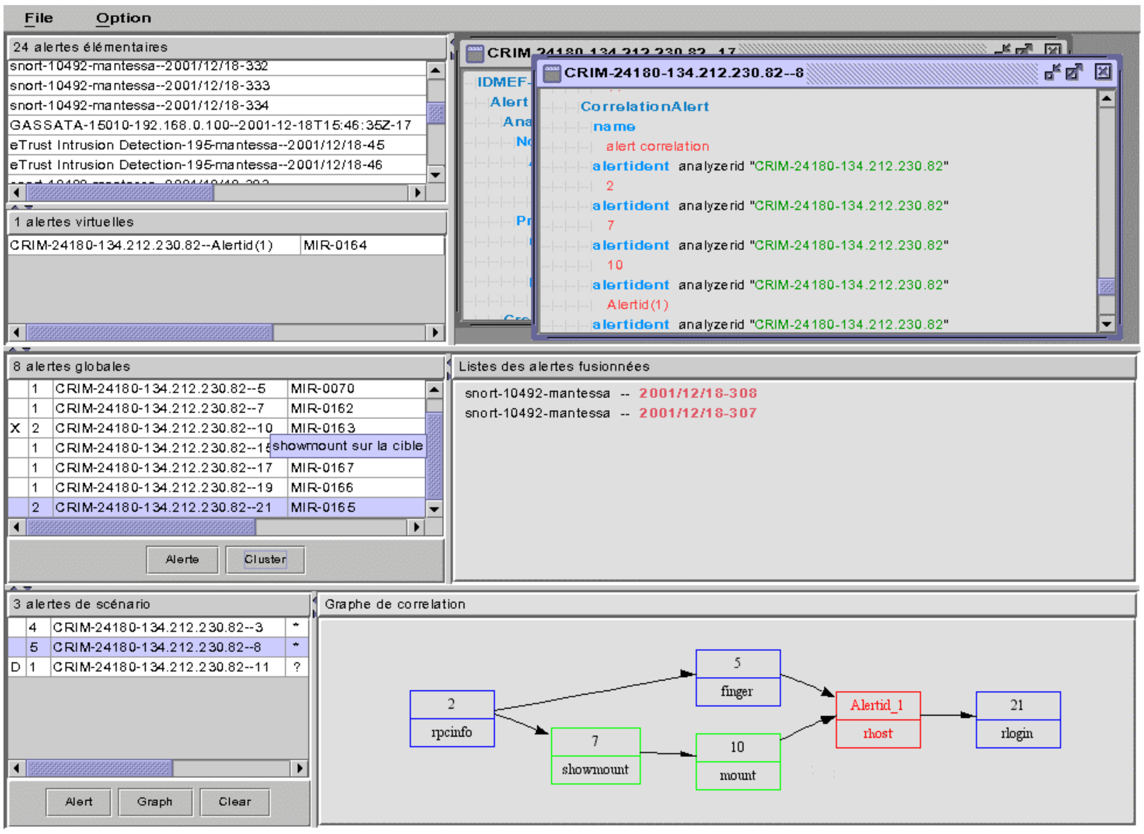
Finally, notice that the architecture we suggest in this paper is centralized, the CRIM module receiving all the alerts generated by the IDS. This is mainly due to technical constraints since we consider in the MIRADOR project that it was not practical to directly create communication between IDS. Such a distributed approach was suggested in [14]. The authors illustrate their approach with the Mitnick attack. There are two steps in this attack. In the first step, the intruder floods a given host H. Then the intruder sends spoofed messages corresponding to H address to establish a communication with a given server S. When S sends an acknowledge to H, H cannot ask to close the connection since it is flooded. In a distributed approach, a first IDS can detect that H is flooded and then asks a second IDS to detect whether S continues receiving messages with H address. If this is the case, then we can conclude that the Mitnick attack is occurring. We agree that this distributed approach is interesting and we plan to analyze it in the future.

Acknowledgements

This work was funded by the DGA/CELAR/CASSI as a part of the Mirador project. The authors would also like to thank all the members of this project: Jacques Capoulade, Mamadou Diop, Samuel Dubus, Aldric Feuillebois (Alcatel CIT), Patrice Carle (ONERA), Ewan Cochevelou, Sylvain Gombault (ENST-Bretagne), Laurent Heye, Ludovic Mé and Cédric Michel (SupElec Rennes).

Annexe 1: Graphical interface

Figure 9 presents a view of CRIM interface. There are 3 sub-windows in this interface. The upper window provides a view of alerts that are directly generated by the IDS connected to CRIM or abduced by CRIM (virtual alerts). The central window corresponds to fusion alerts, that is alerts generated by the merging function of CRIM. Finally, the lower window presents the alerts generated by the online correlation function. It actually shows detection of the “Illegal NFS Mount” scenario presented in Section 5. Alertid_1 is a virtual alert corresponding to attack “MIR-0164” (Modification of .rhost) that is automatically abduced to complete detection of this scenario.



24 alertes élémentaires

```
snort-10492-mantessa--2001/12/18-332
snort-10492-mantessa--2001/12/18-333
snort-10492-mantessa--2001/12/18-334
GASSATA-15010-192.168.0.100--2001-12-18T15:46:35Z-17
eTrust Intrusion Detection-195-mantessa--2001/12/18-45
eTrust Intrusion Detection-195-mantessa--2001/12/18-46
```

1 alertes virtuelles

CRIM-24180-134.212.230.82--Alertid(1)	MIR-0164

CRIM-24180-134.212.230.82--8

```
CorrelationAlert
  name
    alert correlation
  alertident analyzerid "CRIM-24180-134.212.230.82"
    2
  alertident analyzerid "CRIM-24180-134.212.230.82"
    7
  alertident analyzerid "CRIM-24180-134.212.230.82"
    10
  alertident analyzerid "CRIM-24180-134.212.230.82"
    Alertid(1)
  alertident analyzerid "CRIM-24180-134.212.230.82"
```

8 alertes globales

	1	CRIM-24180-134.212.230.82--5	MIR-0070
	1	CRIM-24180-134.212.230.82--7	MIR-0162
X	2	CRIM-24180-134.212.230.82--10	MIR-0163
	1	CRIM-24180-134.212.230.82--15	showmount sur la cible
	1	CRIM-24180-134.212.230.82--17	MIR-0167
	1	CRIM-24180-134.212.230.82--19	MIR-0166
	2	CRIM-24180-134.212.230.82--21	MIR-0165

Listes des alertes fusionnées

```
snort-10492-mantessa -- 2001/12/18-308
snort-10492-mantessa -- 2001/12/18-307
```

3 alertes de scénario

	4	CRIM-24180-134.212.230.82--3	*
	5	CRIM-24180-134.212.230.82--8	*
D	1	CRIM-24180-134.212.230.82--11	?

Graphe de correlation

Figure 9 : Graphical Interface

References

[1] Computer Associates. E-Trust Intrusion Detection. 2000.

[2] D. Curry and H. Debar. "Intrusion Detection Message Exchange Format Data Model and Extensible Markup Language (XML) Document Type Definition". draft-itetf-idwg-idmef-xml-03.txt, February 2001.

[3] F. Cuppens. "Managing alerts in a multi-intrusion detection environment". 17th Annual Computer Security Applications Conference (ACSAC). New-Orleans, December 2001.

[4] F. Cuppens and R. Ortalo. "LAMBDA: A Language to Model a Database for Detection of Attacks". Proceedings of the Third International Workshop on the Recent Advances in Intrusion Detection (RAID'2000), Toulouse, France, October 2000.

[5] M. Diaz. GNU Prolog: A Native Prolog Compiler with Constraint Solving over Finite Domains. Edition 1.4 for GNU Prolog version 1.2.1. http://gnu-prolog.inria.fr/manual/. July, 2000.

[6] H. Debar and A. Wespi. "The Intrusion-Detection Console Correlation Mechanism". Workshop on the Recent Advances in Intrusion Detection (RAID'2001), Davis, USA, October 2001.

[7] M.-Y. Huang. "A Large-scale Distributed Intrusion Detection Framework Based on Attack Strategy Analysis". Proceedings of the First International Workshop on the Recent Advances in Intrusion Detection (RAID'98), Louvain-La-Neuve, Belgium, 1998.

[8] K. Kendall. "A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems". June 1999.

[9] W. Lee. "Combining Knowledge Discovery and Knowledge Engineering to Build IDSs". Proceedings of the Second International Workshop on the Recent Advances in Intrusion Detection (RAID'99), Purdue, USA, October 1999.

[10] R. Lippmann. "Using Key String and Neural Networks to Reduce False Alarms and Detect New Attacks with Sniffer-Based Intrusion Detection Systems". Proceedings of the Second International Workshop on the Recent Advances in Intrusion Detection (RAID'99), Purdue, USA, October 1999.

[11] C. Michel and L. Mé. Adele: "an Attack Description Language for Knowledge-based Intrusion Detection". 16th International Conference on Information Security. Kluwer. June 2001.

[12] M. Roesch. "Snort – Lightweight Intrusion Detection for Networks". Proceedings of USENIX LISA'99, November 1999.

[13] A. Valdes and K. Skinner. "Probabilistic Alert Correlation". Proceedings of the Fourth International Workshop on the Recent Advances in Intrusion Detection (RAID'2001), Davis, USA, October 2001.

[14] J. Yang, P. Ning, X. Wang, and S. Jajodia. "CARDS : A Distributed System for Detecting Coordinated Attacks". IFIP TC11 Sixteenth Annual Working Conference on Information Security, August 2000.

[15] D. Zerkle. „A Data-Mining Analysis of RTID". Proceedings of the Second International Workshop on the Recent Advances in Intrusion Detection (RAID'99), Purdue, USA, October 1999.