Correlation in an intrusion detection process

Frédéric Cuppens[1], Fabien Autrel[1], Alexandre Miège[2] & Salem Benferhat[3]

1: ONERA-CERT, 2 Av. E. Belin, 31055 Toulouse Cedex, France,
2: ENST Paris, 46 rue Barrault, 75014 Paris CEDEX, France,
3: IRIT, 118 route de Narbonne, 31062 Toulouse CEDEX, France
email: {cuppens,autrel,miege}@cert.fr, benferhat@irit.fr

Abstract

Generally, the intruder must perform several actions, organized in an *intrusion scenario*, to achieve his or her malicious objective. We argue that intrusion scenarios can be modelled as a planning process and we suggest modelling a malicious objective as an attempt to violate a given security requirement.

Our proposal is then to extend the definition of attack correlation presented in [2] to correlate attacks with intrusion objectives and to introduce the notion of *anti* correlation. These notions are useful to decide if a sequence of correlated actions can lead to an intrusion objective. This approach provides the security administrator with a global view of what happens in the system. In particular, it controls unobserved actions through hypothesis generation, clusters repeated actions in a single scenario, recognizes intruders that are changing their intrusion objectives and is efficient to detect variations of an intrusion scenario. This approach can also be used to eliminate a category of false positives that correspond to false attacks, that is actions that are not further correlated to an intrusion objective.

1. Introduction

The main objective of computer security is to design and develop computer systems that conform to the specification of a security policy. A security policy is a set of rules that specify the authorizations, prohibitions and obligations of agents (including both users and applications) that can access to the computer system. An intruder (also called hacker or cracker) might be viewed as a malicious agent that tries to violate the security policy. Thus, an intrusion is informally defined as a deliberate attempt to violate the security policy. This intrusion can be an attempt:

- To have an illegal access to some piece of information. In this case, the intrusion violates a confidentiality constraint expressed in the security policy. For instance, sniffing or cracking a password violates a confidentiality constraint saying that the owner of a password must be the only user that knows this password.

- To perform some illegal creation, modification or deletion of some piece of information. In this case, the intrusion violates an integrity constraint expressed in the security policy. For instance, an IP spoofing consists in forging IP packets with illegal address. This is an intrusion that violates an integrity constraint saying that the address of an IP packet must represent the sender of this packet.

- To prevent other users to have legal access to some services or resources. In this case, the intrusion violates an availability constraint expressed in the security policy. For instance, flooding a system with messages so that other users can no longer have an access to this system provides an example of an intrusion against availability.

Notice that sometimes the intruder might perform his intrusion by using a single action. For instance, performing a deny of service using the *ping of death* attack simply requires sending a too long IP packet. However, more complex intrusions generally require several steps to be performed. For instance, let us consider 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 SYN messages corresponding to H address to establish a TCP connection with a given server S. When S sends a SYN-ACK message, H would normally send a RESET message to close the connection. But this is not possible since H is flooded. This enables the intruder to send an ACK message to illegally open a connection with S. Notice also that opening a TCP connection with S is probably not the intruder's final objective. It is likely that the intruder will then attempt to get an access on S for instance by performing a *rlogin*. This means that the *Mitnick* attack will actually represent preliminary steps of a more global intrusion. In the following, we shall call *intrusion scenario* the complete sequence of actions that enables the intruder to achieve his intrusion objective.

Another important point to be mentioned is that the intruder will generally first need to gain knowledge about the target system to be attacked. For instance, let us consider an intruder whose objective is to perform a deny of service (DOS) over the Domain Name Server (DNS) of a given network. In this case, a "brute force" intrusion would be to launch a *Winnuke* attack over all the machines of this network, expecting that the DNS server will be denied at the same time as other machines. However, this is not a very efficient nor clever way to proceed. It is more likely that a careful intruder will first use the *nslookup* command to locate the DNS server and then send a *ping* to check whether this server is active. And if the intruder chooses *Winnuke* to perform the DOS attack, since *Winnuke* only succeeds on Windows machines, this careful intruder will probably check if the DNS server actually supports Windows. For this purpose, the intruder may scan port 139 (NetBios) because NetBios provides good evidence that Windows is active. We shall call *knowledge gathering steps* the set of commands that enables the intruder to gain knowledge about the target system. In our previous example, the knowledge gathering steps correspond to *nslookup*, *ping* and *scan* of port 139. In the following, we shall consider that the knowledge gathering steps are part of the intrusion scenario.

In this context, current intrusion detection technology only detects elementary attacks that correspond to the steps of a given intrusion scenario. They neither provide a global view of the intrusion scenarios in progress nor of the intrusion objectives the intruders attempt to achieve. Therefore, our goal in this paper is twofold. First, we suggest an approach to recognize various steps of an intrusion scenario. We shall call *attack correlation* this first functionality. It is actually an extension of the approach suggested in [2]. Second, when the attack correlation function succeeds in correlating several actions, we want to decide whether the current observations actually correspond to malicious activities or not. We call *malicious intention recognition* this second functionality. Combining these two functionalities would enable the security administrator to have a global understanding of what happens in the system in order to prepare an adequate reaction. Notice also that sometimes, this reaction might be launched before the intrusion scenario is completed, that is before the intrusion objective is actually achieved.

The remainder of this paper is organized as follow. Section 2 introduces preliminary definitions to fix the vocabulary. Section 3 presents our approach to modelling the intrusion process. This model includes a representation of both attacks and intrusion objectives. Our approach is actually derived from planning process models in Artificial Intelligence. These models are used to represent the activity of an agent that attempts to find a sequence of actions that achieve a given objective. We argue that the activity of an intruder who is performing an intrusion scenario is quite similar to a planning process. Section 4 then presents our approach for modelling the intrusion *detection* process. From a formal point of view, this approach uses the same materials as the ones presented in section 3, namely attack and intrusion objective modelling. Based on these materials and following [2], we define the notion of attack and alert correlation and also correlation between an attack and an intrusion objective. We then introduce the notion of anti correlation that is useful to detect that an action disables a given intrusion scenario in progress. Section 5 further refines our approach by introducing abduction in the correlation process. Abduction is used to generate hypotheses. This is useful in two different situations:

1. Abduction of unobserved attacks. This is used to complete detection of an intrusion scenario when some steps in this scenario are not detected (false negatives).

2. Abduction of intrusion objectives. This is useful to anticipate over the intruder intentions when several actions that match an intrusion scenario have been correlated.

Section 6 presents an experimentation of our approach on several examples of intrusion scenarios. Section 7 is a discussion of our approach, compared to other approaches suggested in the literature, in particular approaches based on expert system [7], explicit plan recognition (see for instance [6]) and chronicle recognition [9]. Finally section 9 concludes the paper.

2. Preliminary definitions

In order to avoid any confusion or misunderstanding, and because the intrusion detection vocabulary is not clearly established, we give in this section a brief overview of the terms we shall use in this paper.

Intrusion objective (intrusion detection point of view) An intrusion objective is the final purpose of an intruder, which justifies all its actions. So, from its point of view, the intrusion objective is obvious. By contrast, from the intrusion detection point of view, it is more difficult to determine the possible intrusion objectives and to differentiate them from non malicious activities.

As an intruder aims at committing a forbidden action, we suggest deriving the possible intrusion objectives from the security policy: *any security policy violation is a potential intrusion objective*.

We give three examples corresponding to integrity, confidentiality, and availability violation:

- Objective 1: gaining a non authorized root access right
- Objective 2: having a non authorised read access to a sensitive file
- Objective 3: performing a denial of service attack on the DNS

Malicious action A malicious action enables the intruder to directly achieve an intrusion objective. For instance, thanks to the *Winnuke* attack, an intruder can do a denial of service on a Windows server.

Intrusion scenario As an intrusion objective will often needs several actions to be reached, the intruder needs to draw up an *intrusion scenario*. It is an organised set of actions, which have to be executed following a certain order.
Let us present three intrusion scenarios corresponding to the intrusion objectives described just before.

1. **Illegal NFS mount:** the intruder, say *badguy*, wants to obtain a root access on a target. *badguy* can perform the following actions:[1]
    - *rpcinfo* to know if *portmapper* is running on the target.
    - With the *showmount* command, *badguy* sees the target exportable partitions.
    - *mount* enables *badguy* to mount one of this partition, say the root partition.
    - By modifying the *.rhost* file of this partition, *badguy* gets a root access on the target system.
    - *rlogin* is the final step to access the target system with root privileges.
2. **Illegal file access:** we shall consider the following intrusion scenario example where an unauthorised user *bad_guy* tries to read *secret-file*:[2]

[1] This scenario actually exploits a wrongly configured security policy: the intruder should not be able to mount the root partition.
[2] This is an old intrusion scenario that does no longer work on current UNIX versions but it provides a good example to illustrate various concepts of our approach.

- *bad_guy* creates a file (*touch file*),
- *bad_guy* blocks the printer, by opening the paper trays.
- *lpr -s* enables *bad_guy* to print *file*. With "s" option, the file is not spooled, only its path is saved.
- *bad_guy* deletes *file*
- *bad_guy* creates a symbolic link from *file* to *secret-file*: *ln -s file secret-file*.
- *bad_guy* unblocks the printer and *secret-file* will be printed.

3. **DoS on the DNS:** this intrusion scenario leads to a DoS attack on the DNS server. A possible scenario suggested in the introduction is: *nslookup*, *ping*, *scan* port 139 and *winnuke*.

Action correlation (informal definition) $Action_1$ is correlated with $Action_2$ if $Action_1$ may enable the intruder to then perform $Action_2$.

Suspicious action A suspicious action is defined as an action that can be correlated to a malicious action.

According to this definition, a suspicious action may be an inoffensive action, or may also be a way to execute a malicious action on a following step. For example, scanning port 139 (NetBios) is not a dangerous action. But, if port 139 is open, the intruder knows that Windows is running and can perform the *Winnuke* attack.

Attack An attack is a malicious action or a suspicious action.

This is quite a weak definition of "attack" since it also includes suspicious actions. However, we guess it is close to the intrusion detection terminology since many alerts actually correspond to only suspicious actions. This leads to the following definition of alerts:

Simple alert A *simple alert* is a message sent by an IDS. It results from the detection of a suspicious or a malicious action.

Fusion process and fusion alert The simple alerts generated by different IDS detecting the same attack are merged into a single cluster. This is called *fusion process*. It determines first which are the merging criteria for each type of attack, and then, during the intrusion detection process, uses those criteria to constitute clusters. At last, it generates a *fusion alert* to inform all the security devices of the creation and the content of a new cluster. It is not the purpose of this paper to further present the fusion process but see [1, 10] for different proposals for this process.

Correlation process and scenario alert The *correlation process* receives the fusion alerts and tries to correlate them one by one using correlation rules. When a complete or a partial intrusion scenario is detected, a *scenario alert* is generated. [2] suggests an approach for the correlation process and correlation rules definition. The purpose of this paper is to extend this correlation process.

False positive False positive and false negative are well documented notions in intrusion detection literature. However, regarding false positive we guess it is necessary to distinguish between false detection and false attack.

1. *False detection* corresponds to the occurrence of an alert whereas the corresponding attack did not occur. For instance, this can be due to an IDS weak signature.

2. *False attack* results from detecting a suspicious action that is not further correlated with a malicious action.

We argue that the fusion process is useful to recognize false detection (see [1]). However, it is not sufficient to detect false attacks. For this purpose, the correlation process presented in this paper will be useful.

3. Modelling the intrusion process

The objective of this paper is to detect intrusion scenario and recognize malicious intention. For this purpose, it is first useful to analyse and model how intruders proceed to perform their intrusions.

In our approach the intrusion process is modelled as a planning activity. We assume that the intruder wants to achieve intrusion objectives and, for this purpose, the intruder can use a set of attacks (remember that attacks include both suspicious or malicious actions). In this context, the intruder's problem is to find a sequence of attacks that transform a given initial state into a final state. The initial state corresponds to the system state when the intruder starts his intrusion. And the final state has the property that the intrusion objective is achieved in this state.

We check this approach on several intrusion scenarios, including the three scenarios presented in section 2 but also other scenarios such as the *Mitnick* attack. For every analysed scenario, it was possible to interpret it as a planning process. Due to space limitation, we shall only illustrate this claim on scenario 3 "illegal file access". But before, we need to present our approach to model attacks and intrusion objectives.

3.1. Attack modelling

In the planning context, actions are generally represented by their pre and post conditions. Pre conditions of an action correspond to conditions the system's state must satisfy to perform the action. Post conditions correspond to effects of executing the action on the system's state.

In our model, an attack is similarly represented using three fields: its name, pre condition and post condition. Attack name is a functional expression that represents the name of the attack and its different parameters. Attack pre-condition specifies the logical conditions to be satisfied for the attack to succeed and attack post-condition is a logical condition that specifies the effect of the attack when this attack succeeds.

The pre-condition and post-condition of an attack correspond to 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 system state relevant to the description of an attack. In this language, we assume that terms starting by an upper case letter correspond to variables and other terms are constants.

The 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 conditions of an attack. Another restriction is that negation only applies to predicates, not to conjunctive expressions. The reason of these restrictions will be explained in section 4.

In order to model knowledge gathering actions, we extend language L_1 so that it also includes a meta-predicate (actually a logical modality) *knows*. If A is an agent and p is a formula of L_1, then $knows(A, p)$ means that A knows that p is true. We assume that modality $knows$ satisfies the following axiom for each agent A and formula p: $knows(A, p) \rightarrow p$, that is if A knows that p then p is true.

Figure 1 shows how various steps of scenario *illegal file access* are represented in this model. In this example, we use the following predicates: $file(File)$ ($File$ is a file), $owner(Agent, File)$ ($Agent$ is the owner of $File$), $printer(Printer)$ ($Printer$ is a printer), $blocked(Printer)$ ($Printer$ is blocked), $authorized(Agent, Right, File)$ ($Agent$ has $Right$ access on $File$), $linked(Link, File)$ (there is a logical link from $Link$ to $File$), $queued(File, Printer)$ ($File$ is queued in $Printer$), $read\_access(Agent, File)$

`Action` $touch(Agent, File)$ `Pre:` $true$ `Post:` $file(File), owner(Agent, File)$	`Action` $block(Agent, Printer)$ `Pre:` $printer(Printer),$ $physical\_access(Agent, Printer)$ `Post:` $blocked(Printer)$
`Action` $lpr\text{-}s(Agent, Printer, File)$ `Pre:` $printer(Printer), file(File),$ $authorized(Agent, read, File)$ `Post:` $queued(File, Printer)$	`Action` $remove(Agent, File)$ `Pre:` $owner(Agent, File)$ `Post:` `not` $(file(File))$
`Action` $ln\text{-}s(Agent, Link, File)$ `Pre:` `not` $(file(Link))$ `Post:` $linked(Link, File)$	`Action` $unblock(Agent, Printer)$ `Pre:` $printer(Printer), blocked(Printer),$ $physical\_access(Agent, Printer)$ `Post:` `not` $(blocked(Printer))$
`Action` $print\text{-}process(Printer, Link)$ `Pre:` $queued(Link, Printer),$ $linked(Link, File),$ `not` $(blocked(Printer))$ `Post:` $printed(Printer, File),$ `not` $(queued(Link, Printer))$	`Action` $get\text{-}file(Agent, File)$ `Pre:` $printed(Printer, File),$ $physical\_access(Agent, Printer)$ `Post:` $read\_access(Agent, File)$

Figure 1: Modelling the illegal file access scenario

($Agent$ has a read access on $File$) and $physical\_access(Agent, Printer)$ ($Agent$ has a physical access to $Printer$).

To model the *illegal file access* scenario, we actually specify 8 actions. The 6 first actions $touch(Agent, File)$, $block(Agent, Printer)$, $lpr\text{-}s(Agent, Printer, File)$, $remove(Agent, File)$, $ln\text{-}s(Agent, Link, File)$ and $unblock(Agent, Printer)$ correspond to the various actions performed by the intruder in the *illegal file access* scenario as presented in section 2.

Our model includes two additional actions $print\text{-}process(Printer, Link)$ and $get\text{-}file(Agent, File)$. Action $print\text{-}process(Printer, Link)$ models what happens on $Printer$ when $Link$ is queued: a file is printed if $Printer$ is not blocked. This printed file will be $File$ if there is a logical link between $Link$ and $File$. Action $get\text{-}file(Agent, File)$ corresponds to the physical action performed by $Agent$ to get $File$ after it is printed. This last action actually enables $Agent$ to obtain a read access to $File$.

We argue that these two last actions are necessary to fully represent this scenario. In particular, the intruder has not achieved his intrusion till he has not executed the action get_file. Notice also that the attack will not succeed if $Agent$ has not a physical access to the printer on which the sensitive file is printed.

3.2. Modelling intrusion objective

In our approach intrusion objectives actually correspond to violation of a given security policy. For instance, the security policy may specify (1) that an agent should not gain a root access to a system whereas he is not authorized to do so, or (2) an agent should not obtain a read access to a file whereas he is not authorized to do so, or (3) a DNS system should remain available in any circumstance.

Therefore, an intrusion objective is modelled by a system state condition that corresponds to a violation of the security policy. Figure 2 provides three examples of intrusion objectives that respectively correspond to violation of the three requirements specified in the previous security policy example. For instance, intrusion objective $Dos\_on\_DNS(Host)$ is achieved if $Host$ is a DNS server and there is a DOS attack on this server.

`Intrusion_Objective` $illegal\_root\_access(Host)$ `State_Condition:` $root\_access(Agent, Host)$, `not` $(authorized(Agent, root, Host))$
`Intrusion_Objective` $illegal\_file\_access(File)$ `State_Condition:` $read\_access(Agent, File)$, `not` $(authorized(Agent, read, File))$
`Intrusion_Objective` $DOS\_on\_DNS(Host)$ `State_Condition:` $dns\_server(Host), dos(Host)$

Figure 2: Examples of intrusion objectives

`Domain_rule` $owner\_right(File)$ `Pre:` $owner(Agent, File)$ `Post:` $authorized(Agent, read, File), authorized(Agent, write, File)$
`Domain_rule` $remove\_right(File)$ `Pre:` `not` $file(File)$ `Post:` `not` $(owner(Agent, File))$, `not` $(authorized(Agent, read, File))$, `not` $(authorized(Agent, write, File))$

Figure 3: Examples of domain rules

3.3. Domain rules

In our model, we also include the possibility to specify domain rules. Domain rules are used to represent general properties of system's state through possible relations between predicates. These domain rules are also represented using a pre and post condition but there is a major difference with the pre and post condition of actions. Indeed, an action corresponds to a transition from an initial state to a final state: the pre condition of an action is true in the initial state and the post condition is true in the final state. By contrast, the pre and post conditions of an domain rule are evaluated on the same system state: if the pre condition of an domain rule is true in a given state, then the post condition is also true in the *same* state.

Figure 3 provides two examples of domain rule. Rule $owner\_right(File)$ says that the $Agent$ owner of a given $File$ is automatically authorized to have read and write access to this file. Rule $remove\_right(File)$ says that if $File$ does no longer exist, then there is no longer an owner for this file and read and write access to $File$ are also removed to every $Agent$.

3.4. Planning intrusion scenario

Using the three previous sections, we can now show how the intrusion scenario *illegal file access* is modelled as a planning process. For this purpose, let us consider an intruder, say bad_guy, and a file containing sensitive data, say $secret\_file$. Let us assume that bad_guy wants to achieve the intrusion objective $illegal\_file\_access(secret\_file)$. This means that bad_guy wants to achieve a final system state such that the following condition is satisfied:

- $read\_access(bad\_guy, secret\_file)$, `not` $(authorized(bad\_guy, read, secret\_file))$

Let us also assume that bad_guy starts in the following initial state:

`Step 1:`	$touch(bad\_guy, guy\_file)$
`Step 2:`	$block(bad\_guy, ppt)$
`Step 3:`	$lpr\text{-}s(bad\_guy, ppt, guy\_file)$
`Step 4:`	$remove(bad\_guy, guy\_file)$
`Step 5:`	$ln\text{-}s(bad\_guy, guy\_file, secret\_file)$
`Step 6:`	$unblock(bad\_guy, ppt)$
`Step 7:`	$print\text{-}process(ppt, guy\_file)$
`Step 8:`	$get\text{-}file(bad\_guy, secret\_file)$

Figure 4: Planning the illegal file access scenario

- $file(secret\_file)$, not $(read\_access(bad\_guy, secret\_file))$, $printer(ppt), physical\_access(bad\_guy, ppt)$

That is, in the initial state, $secret\_file$ exists but bad_guy has not a read access to this file and there is a printer ppt and bad_guy has a physical access to this printer.

Now, the planning problem consists in finding a sequence of actions that transforms the initial state into the final state. Figure 4 presents a possible solution to this problem. It is easy to check that objective $illegal\_file\_access(secret\_file)$ is achieved in the state resulting from these 8 steps. Notice that there is another solution that corresponds to starting by blocking the printer and then creating guy_file using the $touch$ command, the other steps being identical to the other solution.

According to our definitions presented in section 2, only $get\text{-}file(bad\_guy, secret\_file)$ corresponding to step 8 is a malicious action since it enables the intruder to achieve the intrusion objective. Steps 1 to 7 are only suspicious actions in the sense that they enable the intruder to then perform step 8.

This might seem surprising since steps 1 to 6 are generally presented as an attack scenario. However, notice that independently each of this step might well correspond to a non malicious action. It is really the combination of these 6 steps that enables the intruder to achieve his objective. However, after step 6, the intruder did not achieve his objective yet. This is why, according to our definition, steps 1 to 6 are only suspicious actions and step 8 is the malicious action. But, of course, if the 6 first steps are observed, we can conclude on the malicious intention of the intruder and it is relevant to react. Actually, it is especially time to react since the intruder perhaps did not get the paper on the printer yet whereas we are quite sure of his malicious intention.

4. Attack and alert correlation

Our approach for intrusion scenario detection uses the same materials as the ones introduced in section 3, namely attack specification through pre and post conditions, intrusion objective corresponding to security policy violation and domain rules. Based on these materials, we shall extend the definition of attack correlation suggested in [2] by defining the notions of objective correlation and *anti* attack and objective correlations.

4.1. Correlation definitions

Let E and F be two logical expressions having the following form:[3]

- $E = expr_{E_1}, expr_{E_2}, ..., expr_{E_m}$
- $F = expr_{F_1}, expr_{F_2}, ..., expr_{F_n}$

[3] Notice that we assume that these two expressions do not include disjunction. This is a restriction which is used to simplify definition of correlation below. Generalising correlation definitions to take into account disjunctions represents further work that remains to be done.

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

- $expr_i = pred$
- $expr_i = \texttt{not}\ (pred)$
- $expr_i = knows(User, pred)$
- $expr_i = knows(User, \texttt{not}\ (pred))$

where $pred$ is a predicate.

Definition 1: Headway correlation We say that logical expressions E and F are headway correlated if the following condition is satisfied:

- there exist i in $[1, m]$ and j in $[1, n]$ such that $expr_{E_i}$ and $expr_{F_j}$ are unifiable through a most general unifier (mgu) Θ.

For instance, the post condition of action $touch(Agent, File)$ is headway correlated with the pre condition of action $remove(Agent, File)$. This is because these two logical expressions have in common predicate $owner(Agent, File)$. After renaming the variables of $owner(Agent, File)$ that respectively appear in the post condition of action $touch$ and the pre condition of action $remove$ into $owner(Agent_1, File_1)$ and $owner(Agent_2, File_2)$, we can conclude that these expressions are unifiable through mgu Θ such that $Agent_1 = Agent_2$ and $File_1 = File_2$.

Definition 2: Knowledge gathering correlation We say that logical expressions E and F are knowledge gathering correlated if the following condition is satisfied:

- there exist i in $[1, m]$ and j in $[1, n]$ such that $expr_{E_i}$ and $knows(User, expr_{F_j})$ are unifiable through a mgu Θ.

For instance, in the *illegal root access* scenario, knowledge gathering correlation applies to correlate the post condition of action $showmount(Agent, Target)$ with the pre condition of action $mount(Agent, Target, Partition)$. Indeed, a possible post condition of action $showmount$ is $knows(Agent, mounted\_partition(Target, Partition))$, that is the $Agent$ performing $showmount$ knows what partitions are mounted on $Target$. On the other hand, $mounted\_partition(Target, Partition)$ appears in the pre condition of action $mount$ and so definition 2 applies.

Definition 3: Correlation We say that logical expressions E and F are correlated if E and F are headway correlated or knowledge gathering correlated.

Definition 4: Direct attack correlation We say that attacks A and B are directly correlated if expressions $Post(A)$ and $Pre(B)$ are correlated.

Intuitively, correlation between attacks A and B means that A enables the intruder to then perform attack B. Figure 5 shows attacks that are directly correlated in the *illegal file access* scenario. In this figure, all variables were renamed to obtain distinct variables in the pre and post conditions of correlated attacks.

Attack A	Attack B	Unifier
$touch(Agent_1, File_1)$	$remove(Agent_2, File_2)$	$Agent_1 = Agent_2, File_1 = File_2$
$remove(Agent_1, File_1)$	$ln\text{-}s(Agent_2, Link_2, File_2)$	$File_1 = Link_2$
$block(Agent_1, Printer_1)$	$unblock(Agent_2, Printer_2)$	$Agent_1 = Agent_2, Printer_1 = Printer_2$
$lpr\text{-}s(Agent_1, Printer_1, File_1)$	$print\text{-}process(Printer_2, Link_2)$	$File_1 = Link_2, Printer_1 = Printer_2$
$ln\text{-}s(Agent_1, Link_1, File_1)$	$print\text{-}process(Printer_2, Link_2)$	$Link_1 = Link_2, Printer_1 = Printer_2$
$unblock(Agent_1, Printer_1)$	$print\text{-}process(Printer_2, Link_2)$	$Printer_1 = Printer_2$
$print\text{-}process(Printer_1, Link_1)$	$get\text{-}file(Agent_2, File_2)$	$File_1 = File_2, Printer_1 = Printer_2$

Figure 5: Direct attack correlation in the illegal file access scenario

Definition 5: Indirect attack correlation We say that attacks A and B are indirectly correlated through domain rules $R_1, ..., R_n$ if the following conditions are satisfied:

- $Post(A)$ is correlated with $Pre(R_1)$ through a mgu Θ_0,
- For each j in $[1, n-1]$, $Post(R_j)$ is correlated with $Pre(R_{j+1})$ through a mgu Θ_j,
- $Post(R_n)$ is correlated with attack $Pre(B)$ through a mgu Θ_n.

For instance, it is easy to verify that attack $touch(Agent, File)$ is indirectly correlated with attack $lpr\text{-}s(Agent, Printer, File)$ through the domain rule $owner\_right(File)$.

Definition 6: Direct objective correlation We say that attack A is directly correlated to intrusion objective O if expressions $Post(A)$ and $State\_condition(O)$ are correlated.

Definition 7: Indirect objective correlation Same definition as definition 5 by replacing $State\_condition(O)$ for $Pre(B)$.

Intuitively, correlation between attack A and intrusion objective O means that attack A enables the intruder to achieve objective O. For instance, attack $get\text{-}file(Agent, File)$ is directly correlated with intrusion objective $illegal\_file\_access(File)$.

4.2. Anti correlation

In this section, we use the same notation as in section 4.1 and define anti correlation as follows:

Definition 8: Anti correlation We say that logical expressions E and F are anti correlated if one of the following conditions is satisfied:

- there exists i in $[1, m]$ and j in $[1, n]$ such that $expr_{E_i}$ and `not` $(expr_{F_j})$ are unifiable through a mgu Θ.
- there exists i in $[1, m]$ and j in $[1, n]$ such that `not` $(expr_{E_i})$ and $expr_{F_j}$ are unifiable through a mgu Θ.

For instance, the post condition of $touch(Agent_1, File_1)$ is anti correlated with the pre condition of $ln\text{-}s(Agent_2, Link_2, File_2)$ through the unifier $File_1 = Link_2$.

Definition 9: Direct anti attack correlation We say that attacks A and B are directly anti correlated if expressions $Post(A)$ and $Pre(B)$ are anti correlated.

In the *illegal access file* scenario, we have the following direct anti attack correlations:

- $touch(Agent_1, File_1)$ and $ln\text{-}s(Agent_2, Link_2, File_2)$ through the unifier $File_1 = Link_2$
- $block(Agent_1, Printer_1)$ and $print\_process(Printer_2, Link_2)$ through the unifier $Printer_1 = Printer_2$

Definition 10: Indirect anti attack correlation We say that attacks A and B are indirectly anti correlated through domain rules $R_1, ..., R_n$ if the following conditions are satisfied:

- $Post(A)$ is correlated with $Pre(R_1)$ through a mgu Θ_0,
- For each j in $[1, n-1]$, $Post(R_j)$ is correlated with $Pre(R_{j+1})$ through a mgu Θ_j,
- $Post(R_n)$ is *anti* correlated with attack $Pre(B)$ through a mgu Θ_n.

For instance, attacks $remove(Agent, File)$ and $lpr\text{-}s(Agent, Printer, File)$ are indirectly anti correlated through domain rule $remove\_right(File)$.

The notion of anti attack correlation is very useful. It enables the correlation process to conclude that some sequences of actions will not succeed and do not correspond to intrusion scenarios. For instance, the sequence:

- $touch(bad\_guy, guy\_file), remove(bad\_guy, guy\_file), lpr\text{-}s(bad\_guy, ppt, guy\_file)$

can be discarded because of the anti correlation between $remove$ and $lpr\text{-}s$. One can also conclude that *illegal access file* scenario will not succeed until the printer is blocked because of the anti correlation between $block$ and $print\text{-}process$.

Similarly, one can define the notions of direct and indirect anti *objective* correlation of an attack A with an intrusion objective O by simply replacing $State\_condition(O)$ for $Pre(B)$ in definition 9 and 10. This is useful to analyse actions that would prevent the intruder to achieve a given intrusion objective. We plan to use this approach to combine the *reaction* process with the intrusion detection process in order to take into account the effect of reaction on the intrusion.

4.3. Generating correlation rules

In [2], we show how to automatically generate *correlation rules*.[4] Due to space limitation, we shall simply give the intuition here.

An attack correlation rule enables the correlation process to correlate two alerts corresponding to correlated attacks. For instance, attack $remove(Agent_1, File_1)$ is correlated to attack $ln\text{-}s(Agent_2, Link_2, File_2)$ when $File_1 = Link_2$. In this case, the associated correlation rule will say that an alert $Alert_1$ corresponding to detection of attack $remove$ can be correlated with an alert $Alert_2$ corresponding to detection of attack $ln\text{-}s$ if the target file associated with $Alert_1$ is equal to the target link associated with $Alert_2$. Of course, an implicit condition to correlate $Alert_1$ with $Alert_2$ is that the attack associated with $Alert_1$ occurred *before* the attack associated with $Alert_2$.

We similarly generate objective correlation rules to correlate an alert with an intrusion objective. When the correlation process receives an alert and there is a correlation rule that applies to correlate this alert to an

[4] These correlation rules may be also specified manually but we argue in [2] that it would be quite tedious for an expert to define accurate correlation rules. Actually, we guess it is one of the main advantages of our approach to automatically derive correlation rules from the specification of elementary attacks.

intrusion objective, the correlation process will check if this objective is achieved or not. For instance, attack $winnuke$ is correlated with objective DOS_on_DNS. If an alert corresponding to attack $winnuke$ on a given $Host$ is received, then the correlation process will check if $Host$ corresponds to a DNS server. Notice that this data is generally not provided in the alert. Therefore, we need to combine "classical" IDS with other tools that collect additional information about the monitored system such as its topology and configuration. This problem is outside the scope of this paper but is further discussed in the conclusion.

5. Abduction in the correlation process

Abductive reasoning consists in inferring plausible explanations from a set of observable facts. It can also be presented as the generation of hypotheses that are missing in a deductive process. In the correlation process, abduction is used in two different situations:

1. When all the steps of a given intrusion scenario are detected, the correlation process will succeed in tracking the intruder by reconstituting his intrusion scenario. However, it may happen that some steps in an intrusion scenario are not detected by any of the IDS. In this case, abduction will try to generate minimal hypotheses corresponding to undetected attacks in order to complete the intrusion scenario detection. In [2], we suggest raising a *virtual alert* for each hypothesis successfully generated.
2. When the correlation process succeeds in correlating several attacks but no intrusion objective is achieved yet, abduction is used to generate an intrusion objective consistent with these attacks. This is used by the correlation process to anticipate over the intruder's intention in order to prepare the most adequate reaction.

5.1. Virtual alerts

The correlation process will attempt to create *virtual alerts* when it is plausible that some attacks are not detected. That is, when two alerts cannot be correlated, we try to insert one or several virtual alerts between them.

Let us describe the two main steps of the virtual alert creation function. Let us assume that $Alert_1$ and $Alert_2$ are not correlated.

- Let $Attack_1$ and $Attack_2$ be the attacks respectively associated with $Alert_1$ and $Alert_2$. The correlation process will then attempt to find a path of attacks to connect $Attack_1$ with $Attack_2$. Currently, the maximal acceptable length of this path is set as a parameter by the security administrator. Of course and this is very important to notice, this path must be formed by attacks that might be not detected by any IDS that provides alerts to the correlation process.

For example, let us assume that, in the illegal root access scenario (see figure 6), the modification of the *.rhost* file is not detected. In this case, the correlation process receive the *rlogin* alert without being able to correlate it with the *mount* alert. However, the correlation process knows that attack *mount* can be correlated with attack *rlogin* through attack *.rhost*.

- The second step of the function replaces the path of attacks by a path of virtual alerts by instantiating each attack. From the first attack we create a first virtual alert. This virtual alert is correlated with $Alert_1$. We do the same for the next attacks of the path until $Alert_2$ is achieved. At this point, the correlation process verifies whether it is possible to correlate the last virtual alert with $Alert_2$.

In the last example, we had a single attack in our path corresponding to the $.rhost$ modification. We create a virtual alert associated with this attack. According to the correlation rules between *mount* alert and *.rhost*

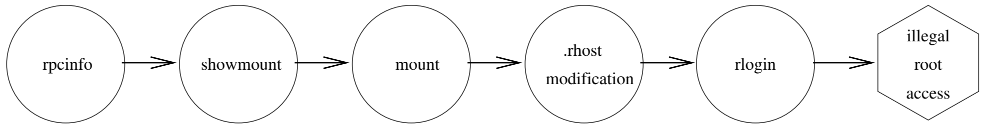

Figure 6: Illegal root access scenario

modification alert, the target IP addresses must be the same. Consequently, the virtual alert is initialised with the target IP address of the *mount* alert. We then check correlation between the virtual alert *.rhost* and the *rlogin* alert. This test could fail if the target IP addresses of these two alerts are not equal.

5.2. Abduction of intrusion objective

When the correlation process has detected the beginning of a scenario, it tries to find out what will be the next steps that might enable the intruder to achieve an intrusion objective.

For this purpose, the correlation process applies an approach similar to the first step used to raise virtual alerts. It analyses the possible correlations between attacks and between an attack and an intrusion objective to find a path of attacks between the last detected alert of the scenario and an intrusion objective.

Of course, it sometimes happens that this alert can be connected to different intrusion objectives through different paths. In this case, our strategy is simply to select an intrusion objective that corresponds to the shortest path. Of course, it would be possible to significantly enhance this simple strategy. This point is further discussed in the conclusion.

6. Examples of scenario detection

In the intrusion detection context, the intruder whose plans are being inferred is not cooperative and observations are gleaned through IDS alerts. This point and the computer security context bring to light several issues to take in consideration. The objective of this section is to show, through the three intrusion scenarios introduced in section 2, how our approach addresses these issues:

- Unobserved actions: There are multiple reasons that can make an attack unobservable. Signature based IDS are not able to recognize unknown exploits and even variations of known exploits can make them undetectable. Furthermore there can be holes in the IDS network coverage that make impossible detection of some malicious attacks.

  Our approach to solve the problem that some steps in an intrusion scenario are not detected is based on abductive reasoning as we show in section 5.

- Repeated observations: it may happen that the intruder will repeat several times the same action, because this is necessary to perform the intrusion scenario or simply because this might be a technique to disrupt the intrusion detection process. In any case, our approach will generate a single alert corresponding to the detection of a single intrusion scenario. For instance, figure 7 shows what will happen if the intruder performs two $rpcinfo$ and then two $showmount$ commands in the *illegal root access* scenario. We see that the result we obtain is more complex than figure 6 but it still corresponds to a single scenario.

- Optional actions: figure 8 shows detection of intrusion scenario DOS_on_DNS when the intruder performs the sequence $nslookup, ping, scan, winnuke$. Actually, actions $ping$ and $scan$ are optional since the intruder may directly attempt the $winnuke$ attack without checking that the server is active (with the $ping$ command) and Windows is installed (with a $scan$ of port 139).

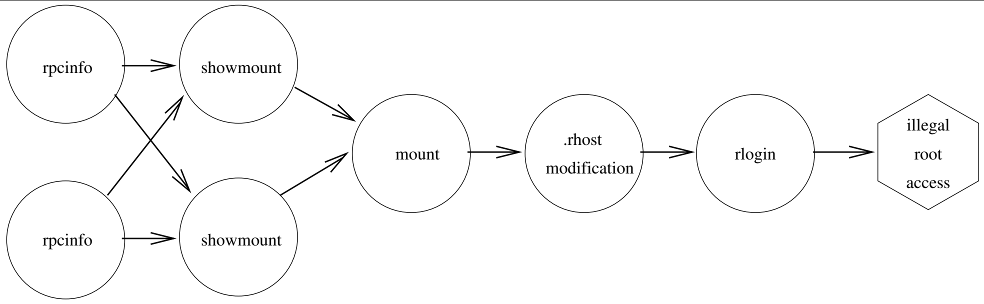

Figure 7: Illegal root access scenario with repeated $rpcinfo$ and $showmount$ commands

Representing intrusion scenario that includes optional actions is immediate in our approach. We have actually nothing to do. If the intruder does not execute $ping$ or $scan$, we shall simply detect a simpler scenario that does not include these actions.

- Representation of intrusion scenario variations: As an intruder executes his plan, his actions may lead him to draw some conclusions i.e. gain some knowledge about the network or some host for example. This may lead in small variations in the execution of the intruder's plan. We have to be able to represent these variations to take them in consideration. For instance, in the DOS_on_DNS scenario, the intruder may prefer using $traceroute$ instead of a $ping$ to know which machines are active in the network he wants to attack.

  Representing intrusion scenario variations is straightforward in our approach. We have simply to specify the pre and post conditions of attack $traceroute$ and the correlation process will automatically derive that replacing $traceroute$ by $ping$ also enables the intruder to achieve the DOS_on_DNS objective.

- Partially ordered plans: By partially plans we point out plans in which the ordering constraint in the actions creates a partial order over the actions. For example in the problem of system scanning, an intruder can collect a large amount of IP addresses and then port sweep each of them or can port sweep each IP as he finds them. This kind of flexible plans is easily detectable in our approach.

- Deactivation actions: there are actions that will disable the possibility to achieve the intrusion objective. In our model, these deactivation actions are detected using *anti* correlation. For instance, figure 9 shows the result we obtain in the case of *illegal access file* scenario. In this figure, dash lines represent anti correlation. As mentioned before, using anti correlation, we can for instance conclude that the sequence $touch, remove, lpr\text{-}s$ does not match an intrusion scenario.

- Inferring unobserved actions from state changes: As noticed in [5], the way IDSs work to detect malicious actions conditions the way these actions are reported. Let us take the example of a service running on a computer. A host based IDS may report that the service has been started but a network based IDS may report only the effects of starting this service. In the first case we observe the action and in the second the state change caused by the action. From the state change we should be able to infer that the service has been started and consider it in our plan recognition task.

  We did not include this point in our approach, but since each attack specification includes a description of its effects (through its post condition), it will be quite straightforward to derive that an action was executed from the observation of a state change. This point is further discussed in the conclusion.

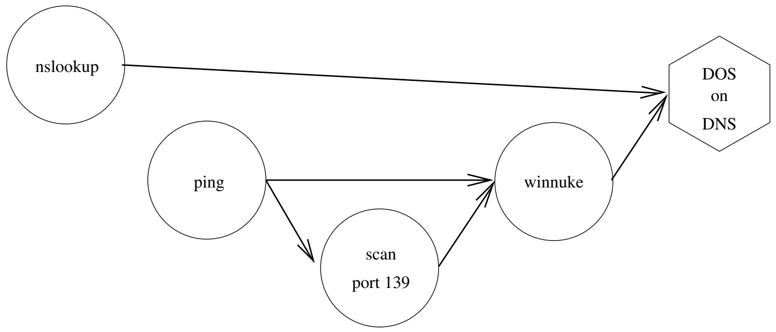

Figure 8: DOS on DNS scenario

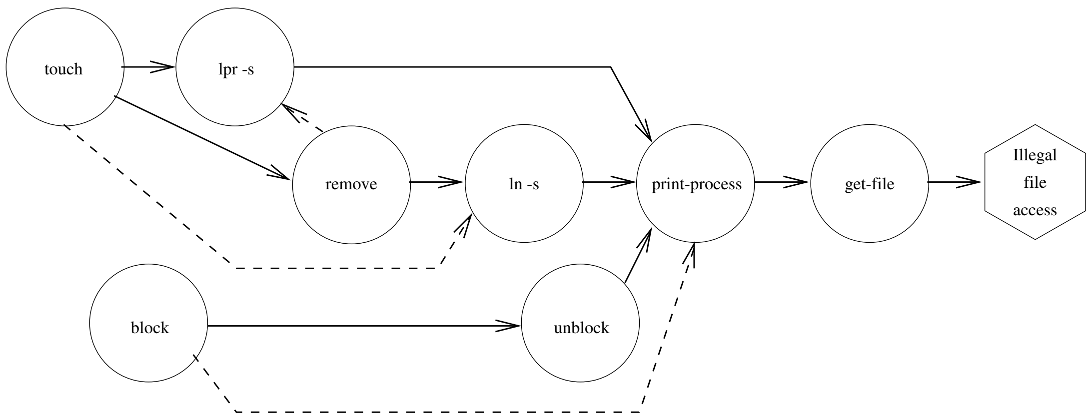

Figure 9: Illegal file access scenario

7. Related works

Several approaches exist in the literature in order to capture an agent's plan, in particular:

- Explicit Plan recognition [5, 6]
- Chronicle recognition [9]
- Expert system approach [8, 7]

7.1. Explicit plan recognition

In the approach suggested by Geib and Goldman in [5], the intrusion scenario detection system must be provided with a *plan library* based on the set of actions the intruder may execute in order to build intrusion scenarios. This represents the major difference with our approach: in our approach, it is neither necessary to explicitly specify such a plan library.

These plans are represented through task decomposition (hierarchical plans). A plan is a graph with goal or top level action as root node and leaves are actions implying this goal. In this approach we can see the problem of plan recognition as finding a minimal set of goals explaining the observed actions. This formalism works as follow: at the beginning the intruder has a set of goals and chooses plans he will execute in order to reach those goals. This set of chosen plans determines all the primitive actions the intruder can execute. The set of pending actions, i.e. the set of actions that can be executed according to the plan hierarchy, is generated as the intruder executes his chosen plans. The pending actions are enabled by the previous actions, more precisely executed actions are removed from the pending set and newly enabled actions are added.

This approach has some advantages:

- Management of unobserved actions: to handle the fact that the attacker's actions are not always detectable, Geib and Goldman construct a set of possible execution traces. This set is built from the execution trace represented by all the observed intruder's actions. After finding all the plans that partially match the observations, the set of possible execution traces is created by adding hypothesized unobserved actions to the execution trace in order to complete it according to the selected plans.
- Abductive reasoning: Geib and Goldman have addressed two problems: inferring unobserved action from observed actions and inferring unobserved actions from state change.

However it has also several drawbacks:

- Exhaustivity of the plan library: since this expert knowledge must be provided by some specialist, it is difficult to assume that the plan library covers all the possible scenarios.
- Risk of explosion of the search space: The fact that hypothesized unobserved actions are inserted in the set of possible execution traces in order to match plans can lead to an explosion of the search space, especially in the case of multiple concurrent intrusion objectives.
- Management of repeated observations: an intruder that will repeat several times the steps of an intrusion scenario will potentially activate the plan recognition several times. This may lead to an explosion of the number of alerts. For instance, if one specifies an intrusion scenario $Winnuke\_Attack$ by the sequence $ping$, $scan$ port 139, $winnuke$ and if the intruder executes 100 $pings$ and 100 $scans$ and then 1 $winnuke$, this may lead to $100 \times 100 = 10000$ detections of the $Winnuke\_attack$.

7.2. Chronicle recognition

The system of chronicle recognition aims at giving an interpretation of the world's evolution giving dated events. It takes in entry a stream of dated events and recognises instances of chronicles as they are developing. It is mainly a temporal reasoning system. It is predictive in the sense that it predicts forthcoming events relevant to its task, it focuses its attention on them and it maintains their temporal windows of relevance. Its main function is to efficiently recognise complex temporal patterns on the fly, as they occur.

Each chronicle can be viewed as a set of events pattern and a set of temporal and contextual constraint over them. If the observed events match the chronicle's patterns and if they occur as the contextual and temporal constraints allow them to, then an instance of the modelled chronicle is recognised. Some hypotheses are done on the events. First, all events specified in a chronicle must be observable, i.e. unobservable activities are not included in the chronicle expression. It is also assumed that the events are reported to the system as they occur and they must be collected in the same order as they occur (synchronization hypothesis).

This approach has several advantages:

- Chronicle based system gives an efficient recognition process.
- The hypothesis stating that all the actions are observable makes unnecessary the abduction of unobserved events.
- The system maintains the set of possible occurring chronicles as the new observations are sequentially collected and treated.
- It is possible to define deactivation events that invalidates a partially recognised chronicle.
- The explosion of the search space is more limited than in the plan recognition approach.

The chronicle main advantages are the consequences of the strong hypotheses made. But the synchronisation hypothesis and the hypothesis made that all the intruder's actions specified in a chronicle can be detected are very hard to fulfil in computer security domain. These hypotheses and the fact that this system is based on a chronicle library point out some drawbacks:

- As for plan recognition systems, the exhaustivity of the plan library is a main concern.
- In computer security domain it is difficult to predict which events will be observable or not.
- Since chronicle recognition assumes that all actions are observable, not including an event because it is sometimes not detected may lead to false positives.
- Including an unobservable event in a chronicle may lead to false negatives.

Actually, we argue that a chronicle system is especially efficient to recognize stereotyped attack scenarios, such as the ones launched by automatic intrusion tools. In this case, it is quite straightforward to represent each attack scenario by a chronicle. We are currently investigating this approach and it will be presented in a forthcoming paper.

7.3. Expert system approach

[7] suggests representing an intrusion scenario as a set of production rules. These rules are fired as the intrusion progresses. This approach is based on the P-Best expert system.

Notice that there is generally not a one to one correspondence between the production rules and various intrusion steps performed by the intruder. Additional production rules are necessary to "control" the intrusion detection process. Actually, starting from the exploit description of an intrusion, the approach requires some adaptations to specify how to detect this intrusion scenario.

This approach has some advantages. In particular, Lindquist and Porras claim that their approach is quite efficient from a performance point of view. The drawbacks of this approach are quite similar to chronicle recognition. The exhaustivity of the production rules is not an easy to solve problem. It also seems that specifying an intrusion scenario must take into account which steps are detected. There is no easy way to abduce unobservable events. Finally, repeated observations will potentially activate detection of the intrusion scenario several times (as in plan or chronicle recognition).

8. Conclusion

Based on the observation that an intrusion scenario might be represented as a planning activity, we suggest a model to recognize intrusion scenarios and malicious intentions. This model does not follow previous proposals that require to explicitly specify a library of intrusion scenarios. Instead, our approach is based on specification of elementary attacks and intrusion objectives. We then show how to derive correlation relations between two attacks or between an attack and an intrusion objective. Detection of complex intrusion scenario is obtained by combining these binary correlation relations.

Our approach is efficient to cluster repeated actions in a single scenario. We also suggest using abduction to recognize intrusion scenarios when some steps in these scenarios are not detected. We then define the notion of *anti* correlation that is useful to recognize a sequence of correlated attacks that does no longer enable the intruder to achieve an intrusion objective. This may be used to eliminate a category of false positives that correspond to false attacks, that is actions that are not further correlated to an intrusion objective.

We have implemented in Prolog the functions that perform attack and objective correlations in the CRIM prototype [1, 2]. Attacks are actually specified in Lambda [3], which is fully compatible with the attack model suggested in this paper and alerts are represented in the IDMEF format [4]. We are currently extending this implementation to include the *anti* correlation functionality.

There are several issues to this work. When the intruder did not achieved his intrusion objective yet but there are several possible intrusion objectives consistent with a given sequence of correlated attacks, our current strategy is simply to select the objective that requires the shortest path of attacks to be achieved. Our course, it would be useful to significantly enhance this strategy. We are currently studying approaches based on Bayesian Network to decide what are the best intrusion objectives that explain all the observations. As suggested in [6], our solution should also able to consider situations where the intruder has multiple goals.

Another point is that to decide if a given intrusion scenario is achieved or not, it is often necessary to combine information provided by "classical" IDS with other information about the system monitored by the IDS: its topology, configuration and other data about the type and version of the operating systems and applications installed in this system. For instance, to decide if the objective DOS_on_DNS is achieved it is necessary to know on which system is installed the DNS server. This kind of data is not provided by classical IDS but other tools exist that may be used to collect it. Since current IDS also provide alerts that do not allow us to distinguish between successful or failing attacks, these additional data would be also useful for that purpose. This problem is currently investigated in the ongoing project DICO.

9. Acknowledgements

This work was partially funded by the DGA/CELAR/CASSI as a part of the Mirador project and then by the French Ministry of Research as part of the DICO project. The authors would like to thank all the members of these projects, especially the members of the sub-project "Correlation": Hervé Debar, Ludovic Mé and Benjamin Morin.

References

[1] F. Cuppens. Managing Alerts in a Multi-Intrusion Detection Environment. In *17th Annual Computer Security Applications Conference New-Orleans*, New-Orleans, USA, December 2001.

[2] F. Cuppens and A. Miège. Alert Correlation in a Cooperative Intrusion Detection Framework. In *IEEE Symposium on Security and Privacy*, Oakland, USA, 2002.

[3] F. Cuppens and R. Ortalo. Lambda: A language to model a database for detection of attacks. In *Third International Workshop on the Recent Advances in Intrusion Detection (RAID'2000)*, Toulouse, France, October 2000.

[4] 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-06.txt, December 2001.

[5] C. Geib and R. Goldman. Plan Recognition in Intrusion Detection Systems. In *DARPA Information Survivability Conference and Exposition (DISCEX)*, June 2001.

[6] C. Geib and R. Goldman. Probabilistic Plan Recognition for Hostile Agents. In *Florida AI Research Symposium (FLAIR)*, Key-West, USA, 2001.

[7] Ulf Lindquist and Philip Porras. Detecting Computer and Network Misuse Through the Production-Based Expert System Toolset (P-Best). In *IEEE Symposium on Security and Privacy*, Oakland, USA, 1999.

[8] A. Mounji and B. Le Charlier. Continuous Assessment of a Unix Configuration: Integrating Intrusion Detection and Configuration Analysis. In *ISOC'97 Symposium on Network and Distributed System Security*, San Diego, USA, February 1997.

[9] M. Ornato and P. Carle. Reconnaissance d'intention sans reconnaissance de plan. In *Journées Francophones d'Intelligence Artificielle Distribuée et Systèmes Multi-Agents*, 1994.

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