Recognizing Malicious Intention in an Intrusion Detection Process

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

1: ONERA Toulouse, 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 [CM02] to correlate attacks with intrusion objectives This notion is useful to decide if a sequence of correlated actions can lead to a security requirement violation. 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 might be viewed as a malicious agent that tries to violate the security policy.

Notice that sometimes the intruder might perform his intrusion by using a single action. However, more complex intrusions generally require several steps to be performed. For instance, there are two steps in the *Mitnick* 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.

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 [CM02]. Second, when the attack correlation succeeds in gathering 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 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. Derived from planning process models in Artificial Intelligence, this model includes a representation of both attacks and intrusion objectives. Section 4 then presents our approach for modelling the intrusion *detection* process. Based on these materials and following [CM02], we define the notion of attack and alert correlation and also correlation between an attack and an intrusion objective. Section 5 further refines our approach by introducing hypothesis generation in the correlation process. This is useful to complete detection of an intrusion scenario when some steps in this scenario are not detected (false negatives). This is also useful to anticipate over the intruder intentions when several actions that match an intrusion scenario have been correlated. Section 6 illustrates our approach on several examples of intrusion scenarios. Finally section 7 concludes the paper.

2 Preliminary definitions

2.1 Intrusion objective and intrusion scenario

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

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 correlated 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, (2) with the *showmount* command, *badguy* sees the target exportable partitions, (3) *mount* enables *badguy* to mount one of this partition, say the root partition, (4) by modifying the *.rhost* file of this partition, (5) *badguy* gets a root access on the target system, (6) *rlogin* is the final step to access the target system with root privileges.

2. **DoS on the DNS:** this intrusion scenario leads to a DoS attack on the DNS server. A possible scenario is: (1) *nslookup* to locate the DNS server, (2) *ping* to check whether the DNS server is active, (3) *scan* port 139 (NetBios) to get evidence that Windows is active on the DNS, (4) *winnuke* that performs a DOS attack on Windows machine.

3. **Illegal file access:** we shall consider the following intrusion scenario example where an unauthorised user *bad_guy* tries to read *secret-file* [ZMB02]: (1) *bad_guy* creates a file (*touch file*), (2) *bad_guy* blocks the printer, by opening the paper trays, (3) *lpr -s* enables *bad_guy* to print *file*, (4) *bad_guy* deletes *file*, (5) *bad_guy* creates a symbolic link from *file* to *secret-file*: *ln -s file secret-file*, (6) *bad_guy* unblocks the printer and *secret-file* will be printed.

2.2 Malicious and suspicious actions

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.

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 or to another suspicious 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.

2.3 Fusion and correlation process

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. The fusion process is not the purpose of this paper; see [Cup01, VS01] for different proposals.

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. The purpose of this paper is to extend this correlation process suggested in [CM02].

3 Modelling intrusion from the intruder point of view

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. The intruder's problem is to find a sequence of attacks that transforms 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. Due to space limitation, we shall only illustrate scenario 3 "illegal file access" as a planning process. But before, we need to present our approach to model attacks and intrusion objectives.

3.1 Action representation

In the planning context, actions are represented by their pre and post conditions. Pre conditions 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. Note that in the modelling of intrusion process system's state is only partially known.

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 *minimal* logical conditions to be satisfied for the attack to succeed and attack post-condition is a logical condition that specifies the *minimal* 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. This means that negation only applies to predicates, not to conjunctive expressions. Taking into account disjunctions is left for future work.

Figure 1 shows how various actions of the example *illegal file access* are represented in this model. To model this scenario, we actually specify 8 actions. The 6 first actions $touch$, $block$, lpr-s, $remove$, ln-s and $unblock$ 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 $print$-$process$ and get-$file$ that usually do not appear in this example. Action $print$-$process(Printer, Link)$ 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-$file(Agent, File)$ 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 $File$.

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

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

3.2 Modelling intrusion objective

An intrusion objective is modelled by a system state condition that corresponds to a violation of the security policy. An intrusion objective contains two fields : its name and a set of conditions denoted by State_condition. 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 $illegal\_file\_access(File)$ is achieved if $Agent$ has a read access to the file $File$ but he is not authorized to read it.

3.3 Domain rules

Domain rules are used to represent general properties of system's state through possible relations between predicates. For convenience matters, these domain rules are also represented using a pre and post condition. However, the pre and post conditions of a domain rule are evaluated on the same system state: if the pre condition of a domain rule is true in a given state, then the post condition is also true in the *same* state.

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

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:

- $file(secret\_file)$, 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.

4 Attack and alert correlation

Our approach for intrusion scenario detection uses the same materials as the ones introduced in section 3. Based on these materials, we shall extend the definition of attack correlation suggested in [CM02] by defining the notion of objective correlation.

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

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

4.1 Correlation definitions

Let E and F be two logical expressions having the form: $E = expr_{E_1}, expr_{E_2}, ..., expr_{E_m}$, and $F = expr_{F_1}, expr_{F_2}, ..., expr_{F_n}$. Each $expr_i$ in E and F is either a predicate or a negation of predicate (not equivalent to $true$), namely $expr_i$ must have one of the following forms: $expr_i = pred$, or $expr_i = \texttt{not}\ (pred)$.

Definition 1: Correlation We say that logical expressions E and F are correlated if 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 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: 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.

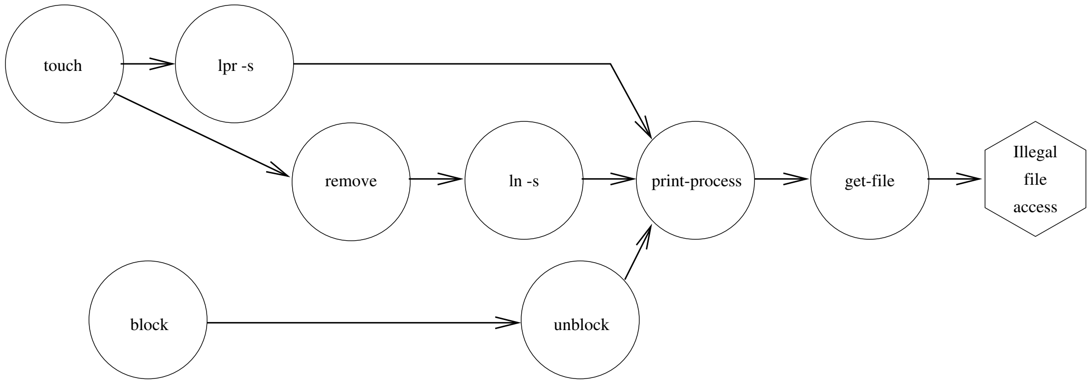

Figure 6: Illegal file access scenario

Definition 3: Indirect attack correlation We say that attacks A and B are indirectly correlated through domain rules $R_1, ..., R_n$ if: (1) $Post(A)$ is correlated with $Pre(R_1)$ through a mgu Θ_0, and (2) For each j in $[1, n-1]$, $Post(R_j)$ is correlated with $Pre(R_{j+1})$ through a mgu Θ_j, and (3) $Post(R_n)$ 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 4: 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 5: 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 $illegal\_file\_access(File)$. Figure 6 shows the result of applying the correlation definitions to the *illegal file access scenario*.

4.2 Generating correlation rules

In [CM02], we show how to automatically generate *correlation rules*. 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 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,

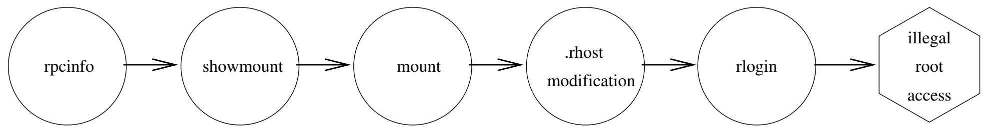

Figure 7: Illegal root access scenario

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 briefly discussed in the conclusion.

5 Hypothesis generation in the correlation process

In the correlation process, hypothesis generation (HG) 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, HG will try to generate minimal hypotheses corresponding to undetected attacks in order to complete the intrusion scenario detection. 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, HG 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 $Attack_2$. Of course, 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[1] (see figure 7), the modification of the *.rhost* file is not detected. In this case, the correlation process receives 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*.

[1]Due to space limitation, we do not give a complete specification of various actions and intrusion objective included in this scenario (see [CM02] for a more complete discussion of this scenario).

- 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* 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 Derivation 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. When this alert can be connected to different intrusion objectives through different paths, our strategy is simply to select an intrusion objective that corresponds to the shortest path.

6 Discussions and 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 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 hypothesis generation as shown in section 5.

- 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). Representing intrusion scenario that includes optional actions is immediate in our approach. If the intruder does not execute $ping$ or $scan$, we shall simply detect a simpler scenario that does not include these actions.

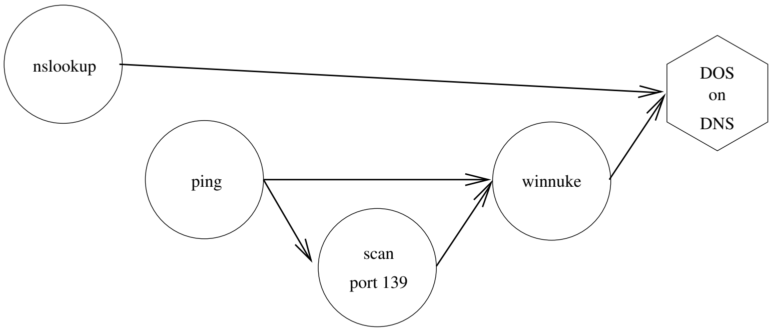

Figure 8: DOS on DNS scenario

- 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.

- 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 as in [GG01a]. 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 $Winnuke\_attack$. Our approach will generate only one alert for such a case, even if this alert corresponds to a complex scenario.

7 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 hypothesis generation to recognize intrusion scenarios when some steps in these scenarios are not detected. We have implemented in Prolog the functions that perform attack and objective correlations in the CRIM prototype [Cup01, CM02]. Attacks are actually specified in Lambda [CO00], which is fully compatible with the attack model suggested in this paper and alerts are represented in the IDMEF format [CD01].

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 studying approaches based on Bayesian Networks to infer the best intrusion objectives that explain all the observations. As suggested in [GG01b], 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 installed applications. 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 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 (see also [MMDD02]).

8 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

[CD01] 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.

[CM02] F. Cuppens and A. Miège. Alert Correlation in a Cooperative Intrusion Detection Framework. In *IEEE Symposium on Security and Privacy*, Oakland, USA, 2002.

[CO00] 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.

[Cup01] F. Cuppens. Managing Alerts in a Multi-Intrusion Detection Environment. In *17th Annual Computer Security Applications Conference New-Orleans*, New-Orleans, USA, December 2001.

[GG01a] C. Geib and R. Goldman. Plan Recognition in Intrusion Detection Systems. In *DARPA Information Survivability Conference and Exposition (DISCEX)*, June 2001.

[GG01b] C. Geib and R. Goldman. Probabilistic Plan Recognition for Hostile Agents. In *Florida AI Research Symposium (FLAIR)*, Key-West, USA, 2001.

[MMDD02] Benjamin Morin, Ludovic Mé, Hervé Debar, and Mireille Ducassé. M2D2: A Formal Data Model for IDS Alert Correlation. In *Fifth International Workshop on the Recent Advances in Intrusion Detection (RAID'2002)*, Zurich, October 2002.

[VS01] 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.

[ZMB02] Jacob Zimmermann, Ludovic Mé, and Christophe Bidan. Introducing reference flow control for intrusion detection at the OS level. In *Fifth International Workshop on the Recent Advances in Intrusion Detection (RAID'2002)*, Zurich, October 2002.