Correlation of Intrusion Symptoms : an Application of Chronicles

Benjamin Morin and Hervé Debar

France Télécom R&D, Caen, France
{benjamin.morin|herve.debar}@rd.francetelecom.com

Abstract. In this paper, we propose a multi-alarm misuse correlation component based on the chronicles formalism. Chronicles provide a high level declarative language and a recognition system that is used in other areas where dynamic systems are monitored. This formalism allows us to reduce the number of alarms shipped to the operator and enhances the quality of the diagnosis provided.

1 Introduction

The diagnosis provided by current intrusion detection systems is spread over numerous fine-grained alarms. As a result, the overall number of alarms is overwhelming. Moreover, their content is so poor that it requires the operator to go back to the original data source to assess the actual severity of the alarms.

Being able to express phenomena involving several alarms[1] is essential in diagnosis applications because using several observations i) strengthens the diagnosis, ii) reduces the overall number of alarms and iii) improves the content of the alarms. Strengthening the diagnosis enables to invalidate or confirm the alarms, which is very important in intrusion detection where false positives are prominent. The number of alarms is reduced because alarms (symptoms) are presented to the operator as labeled groups instead of being presented individually. The content is enhanced because the information of the symptoms are combined. Our approach implies a *multi-event* correlation component using as input IDS alerts.

The correlation component we propose is a *misuse* based. The definition of misuse correlation is similar to misuse intrusion detection : known malicious or undesired sequences of events are searched in the data stream. In our approach, the alarms are checked against a set of multi-events patterns (or signatures) expressed in a dedicated language. Several approaches have been proposed in the research field to provide signatures languages involving many events. In [2], Eckmann *et al* classify languages in six categories : event languages, response languages, reporting languages, correlation languages, exploit languages and detection languages. We are interested here in correlation languages. Correlation

[1] or *events* – in the remainder of this paper, we will either speak of events and alarms because alarms triggered by IDSes are input events of the correlation system

languages rely on alarms provided by IDSes to recognize ongoing attack scenarios. Examples of existing correlation languages are Statl [2], P-Best [4] and Lambda [24].

We propose to use the chronicle formalism proposed by Dousson [11] to correlate alarms. Chronicles are used in many distinct areas [14]. They were primarily designed to analyze sequences of alarms issued by equipments in a telecommunication network and a voltage distribution network. They are now also used in some subtasks of a project aimed at representing car flows in road traffic. In the medical domain, they are being looked at for hepatitis symptoms tracking, intelligent patient monitoring or cardiac arrhythmia detection. We propose to use chronicles to correlate alarms issued by intrusion detection analyzers. Our correlation component uses Dousson's chronicle recognition system (CRS), available at `http://crs.elibel.tm.fr`.

In this paper, we first introduce the chronicles formalism. We then show how chronicles are applied to intrusion detection and illustrate how it solves some intrusion detection issues. We also describe in what extent chronicles integrate with an existing alarm correlation infrastructure. Before concluding and evoking future works, we compare our research with related work.

2 Chronicles

Chronicles provide a framework for modeling dynamic systems. They include an evolution monitoring mechanism to track changes in the modeled system. Recognition of chronicles is based on a formalism in which time is fundamental. This is in contrast with classical expert systems, which base their reasoning on rules, relegating time information to the background.

Chronicles are temporal patterns that represent *possible* evolutions of the observed system. A chronicle is a set of events, linked together by time constraints, whose occurrence may depend on the context. The available time information allows ordering and the specification of time spans between two occurrences of events. In the AI literature, chronicles are related to other approaches such as plan recognition and event calculus (see [12]).

In the remainder of this section, we present the essential features of the chronicles, and briefly sketch the recognition process. Detailed description can be found in [11, 13].

2.1 Representation

In the AI literature, a natural approach to the representation of temporal information consists in associating assertions with particular times. Chronicles representation relies on the *reified temporal logic* formalism [5, 7, 16]. In this formalism, propositional terms are related to times or other propositional terms through additional truth predicates, like *hold*. For example, in a reified logic, one may use $hold(\text{is(light, on)}, T)$ to represent the assertion "light is on over time T".

Time Representation For algorithm complexity reasons, time representation relies on the time points algebra and time is considered as a linearly ordered discrete set of instants whose resolution is sufficient for the environment dynamics.

It should be noticed that in the chronicle formalism, if several *identical* events occur at the same time point, only one is taken into account. As a consequence, the time resolution is very important because in domains like intrusion detection, many identical events may occur within a small time window.

A time interval I is expressed as pair $I = (t_1, t_2)$ corresponding to the lower and upper bound on the temporal distance between two time points t_1 and t_2.

Domain Attributes In the reified logic formalism, the environment is described through domain attributes. Domain attributes are the atemporal propositions of the modeled environment.

A domain attribute is a couple $P(a_1, \ldots, a_n) : v$, where P is the attribute name, $a_1, \ldots, a_n$ its arguments and v its value. For example, $Load(host)$ can be a measure of a server load, and the possible values `{low,medium,high}`. Special attributes, called *messages*, are attributes without any associated value.

Reifying Predicates Reifying predicates are used to temporally qualify the set of domain attributes. Their syntax and informal semantics are sketched in Figure 1. The predicates used in chronicles are *hold*, *event*, *noevent* and *occurs*.

$hold(P : v, (t_1, t_2))$	The domain attribute P must keep the value v over the interval $[t_1, t_2[$.
$event(P : (v_1, v_2), t)$	The attribute P changed its value from v_1 to v_2 at t.
$event(P, t)$	Message P occurs at t.
$noevent(P, (t_1, t_2))$	The chronicle would not be recognized if any change of the value of the domain attribute P occurs between t_1 and t_2.
$occurs((n_1, n_2), P, (t_1, t_2))$ $(0 \leqslant n_1 \leqslant n_2)$	the event that matches the pattern P occurred exactly N times between the two time points t_1 and t_2, and $n_1 \leqslant N \leqslant n_2$. The value ∞ can be used for n_2.

$$occurs \text{ is unifying because } \begin{cases} noevent(P, (t_1, t_2)) \equiv occurs((0, 0), P, (t_1, t_2)) \\ event(P, t_1) \equiv occurs((1, \infty), P, (t_1, t_1 + 1)) \end{cases}$$

Fig. 1. Reifying Predicates

- The *hold* predicate models chronicle assertions (assertions for short). Assertions represent persistence of the value of a domain attribute over an interval, without knowing when this value was reached.
- The *event* predicate expresses a time stamped instance of a pattern. An event has no duration. Events denote a change of the value of a domain attribute.

– The *noevent* predicate expresses forbidden events, i.e. events whose occurrence leads to the invalidation of a chronicle instance during the recognition process.
– The *occurs* is a counting predicate.

Chronicle Model A chronicle model (or *chronicle*) represents a piece of evolution of the world. Chronicles are made of i) a set of time points, ii) a set of temporal constraints between the time points, iii) a set of event patterns which represent relevant changes of the world for this chronicle, iv) a set of assertions patterns which represent the context of the occurrences of events, and v) a set of external actions which will be performed by the system when a chronicle is recognized. Actions are not limited to report generation : the system can generate events and assertions. Both of them can later interact with other chronicles instances. Reinserting previously recognized chronicles in the flow of input events is referred to as "looping" functionality in the remainder (see section 3.4).

Chronicle models are expressed in the chronicle language. After a compilation stage during which the consistency of the chronicle constraints is tested, the chronicles are coded into efficient data structures used for the recognition process described thereafter.

2.2 Chronicle Recognition

After the chronicle models compilation, the recognition system is initialized by creating an empty chronicle instance for each chronicle model. A chronicle *instance* is a chronicle for which a complete match is not found yet. The chronicle recognition system then processes the stream of input events in one shot and on-line.

An event whose atemporal state unifies with a pattern of a chronicle is always considered for integration in a chronicle ; the integration solely depends on the suitability of the chronicle temporal constraints, the previously integrated events and the event's timestamp. Events may be shared by many chronicles and the system is able to manage all the concurrent instances. The recognition process manages a set of partial instances of chronicles as a set of time windows (one for each forthcoming event) that is gradually constrained by each new matched event.

An event occurrence may also lead chronicle instances to be destroyed because an expected event's deadline is reached, and so all chronicles waiting for any event before this deadline are destroyed. Outdated assertions can also be suppressed after an event occurence.

If an assertion is violated or if a deadline expires, then a chronicle instance is destroyed.

When integrating an event occurence in a chronicle instance, the system cannot *a priori* be sure that the event will integrate well in the chronicle with regard to the forthcoming events. It is not possible to integrate an event inside a chronicle without maintaining the hypothesis that it is not necessarily *this* chronicle instance that will be recognized. As a result, every chronicle instance

is duplicated *before* the integration of an event. The systems maintains parallel hypothesis so that *all* event sequences satisfying the constraints are recognized.

```
1  chronicle example1 {
2    event(e1,t1);
3    event(e2,t2);
4    event(e3,t3);
5
6    t1<t2<t3
7    t3-t2 <= 4
8  }
```

Fig. 2. A Chronicle Example

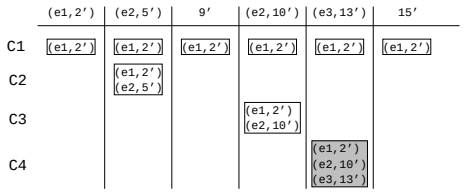

Fig. 3. Duplication example

We illustrate duplication of chronicles with the example in Figure 3. Let us consider the following chronicle :

$$event(e_1, t_1) \wedge event(e_2, t_2) \wedge event(e_3, t_3) \wedge (t_1 < t_2 < t_3) \wedge t_3 - t_2 \leqslant 4$$

which is equivalent to the one represented in the chronicle language in Figure 2.

The event stream is made of e_1 at 2', followed by e_2 at 5' followed by another e_2 at 10' and a e_3 at 13'. When e_1 arrives, a chronicle instance (C_1) of chronicle model C is created. When e_2 arrives, a duplicate of C_1 is created (C_2), and e_2 is integrated in C_2. At 9', C_2 dies because the constraint $t_{e_3} - t_{e_2} \leqslant 4$ is not true anymore. When the second e_2 arrives, a duplicate of C_1 (C_3) is created. When e_3 occurs , a duplicate of C_3 (C_4) is created and the chronicle is recognized (shaded box on figure). At 15', C_3 dies.

This mechanism imposes the chronicle recognition system to be exhaustive, i.e. all the possible instances of the defined chronicles are identified by the system.

For example, if we consider the chronicle

$$event(a, t_1) \wedge event(b, t_2) \wedge event(c, t_3) \wedge t_2 < t_3$$

and the event stream [2],

$$a_1, a_2, b_1, c_1, a_3$$

then the chronicle is recognized three times : $\{a_1, b_1, c_1\}$, $\{a_2, b_1, c_1\}$, $\{a_3, b_1, c_1\}$.

Chronicles duplication imposes chronicle models to be written with care. As a matter of fact, if no chronicle invalidation mechanism is specified in a chronicle model, the chronicle instances tree may grow up indefinitely because of chronicles living forever. Chronicles may be invalidated either with an assertion violation or a deadline being reached. As a result, in order to prevent chronicle instances to live forever, assertions (like *hold* or *noevent*) and/or quantitative time constraints (like $(t_2 - t_1) < 2$) should be specified inside chronicle models.

When a complete match is found, a chronicle is *recognized*, and the associated action is performed by the system.

3 Using Chronicles to Correlate Intrusion Alarms

The current three major issues in intrusion detection are alarm overload, poorness of the alarms semantics and false negatives. In our approach, we explicitly address the first two. The false negative issue is partly solved by making complementary sensors cooperate to provide an appropriate coverage of the monitoring of the environment. Cooperation is a kind of correlation that involves fusion of redundant alarms and synthesis of complementary alarms, and can be achieved by chronicles because we are not restricted to using a single input stream. Cooperation is indeed all the more essential as the multiplication of analyzers also multiplies the alarms. However, contrary to Cuppens in [24] whose correlation process infers unobserved alarms from attack scenarios, we solely rely on *available* alarms. We do not generate *almost recognized*[3] chronicles because this would imply that optional events are used inside chronicles models which could consequently be removed.

In the remainder of this section, we first briefly discuss the informal semantics of the chronicles used in intrusion detection. Then, we give examples to illustrate how chronicles can be used to enhance the content of the alarm and reduce the amount of alarms.

The domain attribute used in the following chronicle models is a triplet $alarm(name, src, trg)$ where $name$ is the attack identifier ($e.g$ "`cmd.exe access`"), src is the attack source ($e.g$ an IP address) and trg is the attack target ($e.g$ an IP address). In fact, the *alarm* term may denote real attacks or benign events. Using the chronicle language, the *alarm* domain attribute is declared as follows :

```
message alarm[?name,?src,?trg]
{
}
```

[2] indices are only used to distinguish event instances and we do not provide timestamps because we do not need them for the example

[3] i.e chronicles whose expected event set is not complete

Constraints on the parameter values can be specified inside the brackets. The `?` is used to inform the system that attributes are variables that should be instantiated by the chronicle recognition system when an event occurs.

3.1 Informal semantics of the chronicles applied to intrusion detection

Chronicles model phenomena which involve more than one event. This definition does not presume the semantic of a chronicle. Actually, in the intrusion detection context, the modeled phenomena may either be *normal* or *malicious*. In this section, we describe these two kinds of chronicles.

Normal phenomena False alarms (false positives) are the primary cause of alarm overload. Although many false positives could be avoided by using more sophisticated signatures and detection mechanisms, it should be noted that some attacks can only be characterized by a single event. As a result, legitimate actions can be confused with attacks.

To solve this, it is possible to discriminate legitimate actions from attacks instead of discriminating attacks from legitimate actions. In this case, chronicles represent *normal* phenomena which involve an alarm as well as other peripheral and innocuous events which are indicative of normal activity. Paragraph 3.2 illustrates this situation. The recognition of such a chronicle invalidates the alarm ; a chronicle invalidation (i.e a chronicle which is not recognized) means that the alarm is indicative of a real attack. In the former case, the alarm is not directly shown to the security operator (it is included in a recognized chronicle) ; in the latter case, the alarm is directly provided to the operator.

Notice that a chronicle instance can also be invalidated because only innocuous events are observed, but no alarms. In this case, the innocuous events shall not be shown to the operator. Examples of innocuous events are provided in Section 3.2.

Malicious phenomena Some attacks are characterized by several suspicious events. In this case, sensors trigger one alarm per event. Chronicles can be used to model these phenomena. Such a chronicle recognition contributes to alarm reduction because only one alarm (the recognized chronicle) is provided to the operator instead of each individual alarm. It also contributes to the semantic improvement because the recognized phenomena is more significant than each individual alarm. If partial chronicle are invalidated, their constitutive alarms are provided to the operator individually and may be used in other correlation mechanisms.

Innocuous events used in this kind of chronicles shall not be provided to the operator if the chronicle is invalidated. If a chronicle is recognized, then the involved innocuous events are available to the operator for investigation.

In the intrusion detection literature, alarm correlation often refers to *attack scenarios*. An attack scenario is a sequence of *explicit* attack steps which are

logically linked and lead to an objective. A portscan followed by a buffer overflow against a given service is an example of attack scenario.

An attack scenario can be modeled by a chronicle. However, we do not intend to use chronicles to model attack scenarios. There are two reasons for this. Firstly, the relevance of an attack scenario is questionable because many (unpredictable) paths may lead to a given attack objective. Secondly, it is hard to specify quantitative time constraints in chronicles because the time gaps between each step may vary a lot, depending on how hurried the attacker is. The attacker may even work on time gaps to evade detection.

As a result, malicious phenomena modeled with chronicles are phenomena whose occurrences are deterministic. Examples of such phenomena are given in sections 3.4 and 3.3.

3.2 Alarm semantics improvement

```
1  chronicle shellcode_mitigation[?source, ?target]
2  {
3     event(alarm[ftp_retr_request,?source,?target], t1)
4     event(alarm[shellcode,?source,?target], t2)
5     noevent(alarm[ftp_transfer_complete,?target,?source],
6                                                 (t1+1,t3-1))
7     event(alarm[ftp_transfer_complete,?target,?source], t3)
8
9     t1<t2<t3
10
11    when recognized {
12      emit event(alarm[shellcode_mitigation, ?source, ?target], t2);
13    }
14 }
```

Fig. 4. A Chronicle Example : Alarm Mitigation

Description of the Phenomenon False alarms are the main cause of alarm overload ; Julisch reports that they represent up to 99% of the overall number of alarms [19].

We believe that the diagnosis provided by intrusion detection systems can be strengthened by taking into account contextual events to discriminate true from false positives. Contextual events can be benign events whose occurrence can reinforce or mitigate the confidence an operator has in an alarm. In that sense, chronicles can both represent known false positive cases and true positives.

To illustrate this, we propose a chronicle which is used identify a recurrent false positive triggered by a network IDS in our network, pretending that buffer

overflow attacks occur. The shellcodes used in some buffer overflow attacks contain long `0x90` bytes strings. Many misuse network IDSes signatures are based on this property to detect buffer overflow attacks in a generic way. However, this kind of signature can provoke false positives because legitimate binary data going on the wire can match the signature. This is the case with ftp file transfers : binary file transfers can trigger alarms because the probability for a binary file to contain `0x90` bytes strings is potentially high[4]. Deactivating the signature is not a solution because true attacks against ftp servers would not be detected anymore.

One solution can consist in mitigating the alarm severity when it is triggered during a file transfer, *i.e* between a request from a client and the end of the file transfer. This implies that sensors generate events for each file retrieval request and the corresponding acknowledgement message. As a result, every file transfer provokes two innocuous events. Notice that the frequency of file transfer requests is moderate with regard to the events throughput managed by chronicles, so these innocuous events do not burden the recognition process. However, a security operator should keep this consideration in mind when writing chronicles.

Description of the Chronicle The corresponding chronicle is in Figure 4. The goal of this chronicle is to generate a report informing the operator that a buffer overflow alarm was raised, but it occurred during a file transfer, so it is probably a false alarm. As a result, any buffer overflow alarm that is not inside a chronicle is really suspect.

In this chronicle, `ftp_retr_request` and `ftp_transfer_complete` are the innocuous events which respectively indicate a FTP file transfer request made by the client and the end of file transfer. The `shellcode` alarm is the actual buffer overflow attempt. A sensor is required to trigger the first two events. Snort can be used for that purpose, with the adequate signatures to monitor control commands of the FTP protocol.

The order in which the reifying predicates are specified does not matter because the system relies on the temporal constraints.

The temporal symbols `t1`,`t2` and `t3` are variables which are instantiated by the system. Note that contrary to the domain attributes parameters, temporal symbols do not need to be prefixed by a "?", since temporal symbols only denote variables (absolute dates cannot be used as time symbols). The chronicle recognition system instantiates `t1` (resp. `t2` and `t3`) with the `ftp_retr_request` (resp. `shellcode` and `ftp_transfer_complete`) event timestamp.

The use of identical variable names as parameters of the domain attributes implicitly imposes the source and target involved in the chronicle to be identical.

The *noevent* predicate in line 5 is necessary to prevent chronicles to live forever in the system (see section 2.2). No quantitative temporal constraint is specified in this chronicle (we do not know how long a file transfer may last) and since the CRS recognition is exhaustive, the `ftp_transfer_complete` event could be used as the end of an earlier chronicle instance being recognized ; as a

[4] http://www.whitehats.com/info/IDS181

result, we need to add a constraint saying that `ftp_transfer_complete` alarms should not occur twice within a chronicle.

When faced with a "normal" ftp transaction, (*i.e.* a `ftp_retr_request` followed by a `ftp_transfer_complete` event), the chronicle recognition system discards a chronicle instance when receiving the `ftp_transfer_complete` event, because the chronicle constraints (`t2<t3`) cannot be satisfied anymore.

3.3 Alarm reduction

```
1  chronicle portscan[?source, ?target]
2  {
3     event(alarm[sid_1,?source,?target], t1)
4     occurs((1,+oo),alarm[sid_2,?source,?target], (t1+1,t2))
5     noevent(alarm[sid_3,?source,?target], (t1,t2))
6     event(alarm[sid_3,?source,?target], t2+1)
7     t1<t2
8
9     when recognized {
10      emit event(alarm[portscan, ?source, ?target], t2);
11    }
12 }
```

Fig. 5. A Chronicle Example : portscan detected by Snort

Description of the Phenomenon Intrusion detection systems tend to spread their diagnosis over many alarms, mainly because the analysis is performed on single events ; as a result, alarms are too fine-grained : a single phenomenon involving many events –be it benign or not– provokes many alarms. Let us take the example of portscan detection by Snort to illustrate this.

When detecting portscans, Snort generates three kinds of alarms : a `portscan_begin` alarm, several `portscan_status` during the scan, and finally a `portscan_end` when the portscan is supposedly finished.

Description of the Chronicle A simple yet effective chronicle model to synthesize portscan alarms is provided in Figure 5. The `sid_1`, `sid_2` and `sid_3` alarms respectively correspond to the `portscan_begin`, `portscan_status` and `portscan_end` alarms. These alarms should have the same source and the same target. The first event (line 1) initiates the chronicle and instantiates t1; t2 is instantiated by the last event (line 6) ; between t1 and t2+1, an infinite number of `portscan_status` alarms may occur.

Portscans are recurrent phenomena. For the same reason as the previous chronicle example, we need to add a constraint saying that portscan end alarms should not occur twice within a chronicle.

3.4 Alarm semantics improvement and alarm reduction

```
1  chronicle nimda[?source, ?target]
2  {
3    occurs((1,2),alarm[iis_code_red_ii_root_exe,?source,?target],
4                                                          (t,t+2000))
5    occurs((1,4),alarm[iis_decode_bug,?source,?target],(t,t+2000))
6    occurs((1,14),alarm[iis_cmd_exe,?source,?target],(t, t+2000))
7    occurs((1,3),alarm[web_dot_dot,?source,?target],(t,t+2000))
8    occurs((1,2),alarm[iis_unicode,?source,?target],(t,t+2000))
9    occurs((1,1),alarm[iis_unicode2,?source,?target],(t,t+2000))
10   occurs((1,1),alarm[iis_unicode3,?source,?target],(t,t+2000))
11   occurs((1,1),alarm[iis_decode_bug3,?source,?target],(t,t+2000))
12   occurs((1,1),alarm[iis_decode_bug2,?source,?target],(t,t+2000))
13   occurs((1,1),alarm[iis_decode_bug4,?source,?target],(t,t+2000))
14
15   when recognized {
16      emit event(alarm[nimda, ?source, ?target], t);
17   }
18 }
```

Fig. 6. A Chronicle Example : a Nimda worm attempt detected by Dragon

Description of the Phenomenon Recognizing known phenomena in which many events are involved both enables reduction of the number of alarms (we only have to consider the alarm set) and semantic enhancement (the identified phenomenon). More and more attacks are automated processes, making it possible to write interesting chronicles because the intrusion steps are always the same.

Example of such phenomena are worms. Worms attacks involve many events, each of which can trigger one or more alarm. As worms are recurrent attacks, reducing the number of alarms for each attempt has a strong impact on the overall alarm excess.

The Nimda worm attacks vulnerable IIS web servers. During each infection attempt, many suspect or malicious HTTP requests are sent to the target. Thus, each Nimda attempt triggers many alarms by conventional IDSes : Snort[5] generates about 20 alarms ; Dragon[6] generates about 30 alarms.

[5] http://www.snort.org
[6] http://dragon.enterasys.com

Description of the Chronicle In Figure 6 we show a chronicle suited for Dragon alarms. A Nimda attempt is characterized by a burst of 10 distinct alarms. Each alarm can occur several times (for instance, the `iis_cmd_exe` occurs 1 to 14 times at each attempt). All the alarms should have the same source and the same target, and they all occur within a 2s time window (the resolution here is 1 ms). When such an alarm burst occurs, a synthetic alarm `nimda_worm_attempt` is reported. Only one synthetic alarm represents the 30 original ones.

```
1  chronicle new_infection[?victim, ?attacker]
2  {
3    event(infected[?victim, somebody]:(true, false), t0)
4    noevent(infected[?victim, ?]:(false, true), (t0,t1))
5
6    event(alarm[nimda, ?attacker, ?victim], t1)
7    event(alarm[nimda, ?victim, ?], t2)
8
9    t2 - t1 in [1000,10000]
10   when recognized {
11       emit event(infected[?victim, ?attacker]:(false,true), t1)
12   }
13 }
```

Fig. 7. Identifying new Nimda infections

We provide an complementary chronicle example called `new_infection` in Figure 7. This example illustrate two important things : the chronicles *looping* functionality and the use of domain attributes which are not alarms. The goal of this chronicle is to recognize every new server infection by the Nimda worm. The first parameter of the `new_infection` chronicle is the newly infected server and the second parameter is the host causing the infection.

We can say that a host b is infected by a if a Nimda attempt from a to b is detected (line 6), followed by Nimda attempts from b to any host (line 7) 1 to 10 seconds later (line 8).

The problem is that a host still receives attack attempts although it is already infected. As a result, if we only use the previous two patterns to detect newly infected hosts, the chronicle would be recognized every time b is attacked. We want the chronicle to be recognized only once, so we need to *tag* infected hosts : b is newly infected only if it was not infected before the attack from a.

We use a *infected(?victim, ?attacker)* $\in \{true, false\}$ domain attribute. Note that *infected* is not an alarm. It is rather a topology-like relation. Note also that contrary to alarms which are messages, *infected* is a valued domain attribute.

Line 3 of the chronicle is the initialization of the state of the hosts : the chronicle recognition system should receive events saying that by default, no host is infected ("`victim` is infected by `somebody`" is false).

Line 4 should be read as "[For the chronicle to be recognized], `victim` must not have been previously infected by *any* host". When used solely, the "?" symbol represents a variable whose value should not be instantiated (*any*).

Line 6 and 7 are the manifestations of a successful infection. When the chronicle is recognized, the status of `victim` is updated (`infected[?victim,?attacker]` becomes true when the attempt occurs – see line 11).

3.5 Sensor cooperation

Description of the Phenomenon The chronicle language is a high level declarative language. It does not presume the nature of the underlying input event stream. As a result, cooperation (*i.e.* correlation of alarms from heterogenous sources) is naturally modeled with chronicles. Of course, the input format must be compliant with the one expected by the chronicle recognition system (the domain attributes properties –arity, parameters domain values– have to be defined).

The upstream sensors that generate alarms must have their clocks synchronized, and the gap between two clocks should be coherent with the time resolution used in the chronicle models. If these conditions hold, then chronicles are able to manage delays due to sensors processing or transport durations. Whether an event is delayed or not, its integration in a chronicle is done in the same way because the chronicle temporal reasoning is solely performed over the events timestamp, not on the running clock.

```
1  chronicle successful_codeExec[?source, ?target]
2  {
3     event(alarm[?bufov_snort_alarm, ?source, ?target], t1);
4     event(alarm[shell_exec, ?, ?target], t2);
5     noevent(alarm[login, ?, ?target], (t1,t2));
6     ?bufov_snort_alarm in {sid_203, sid_34}
7
8     t2 - t1 in [0,100]
9
10    when recognized {
11      emit event(alarm[successful_buffer_overflow, ?source, ?target],
12                                                                  t2);
13    }
14 }
```

Fig. 8. A Chronicle example : example of IDS cooperation

Description of the Chronicle In the example provided in Figure 8, three sensors are used : Snort, Snare and Syslog. The Snort sensor sends the alarm `?bufov_snort_alarm`, `shell_exec` is triggered by a system-based monitoring tool (Snare) and the `login` alarm is sent by Syslog. The chronicle monitors successful attacks resulting in the execution of a shell on a host.

In line 3, we wait for any alarm about a buffer overflow attack. In line 6, the Snort alarm names (`sid_xxx`) that refer to buffer overflow attacks are enumerated. If such an attack is followed by a shell execution on the same target, and no login execution occurred between the two (which could justify the shell execution), then the chronicle is recognized.

3.6 Experimental Results

The chronicle models proposed in this paper have not been tested on a live system yet. We plan to do this in the near future. They have only been validated on alarm logs collected in our network.

However, concerning the ability of the chronicle recognition system to cope with intrusion alarm flow, one should notice that chronicles were primarily designed to diagnose failures in telecommunication network by analyzing alarms issued by equipments. In this context, the recognition system must be efficient because it has to deal with alarm bursts and high alarm rates.

The alarm rates observed in the intrusion detection field are of the same order as the ones observed in other fields where the chronicles have already successfully been applied.

The performance issue may arise if no chronicle invalidation mechanism is specified by the chronicle writer in a chronicle model. In this case, the chronicle instances tree may grow up indefinitely because of chronicles living forever (see section 2.2).

Performance also depends on the number of chronicle models used in parallel. The time required to process an event grows linearly with the number of chronicle models. In [11], efficiency experiments are performed with 80 chronicle models, containing about 10 event patterns among 50 domain attributes, 20 temporal constraints and 4 assertions. In this configuration, the integration of an event required about 10 ms. Detailed efficiency arguments about chronicles can be found in [11] (pages 78–81).

4 M2D2 : An Intrusion Alarm Correlation Infrastructure

Except the chronicle model proposed in Figure 7, $alarm(name, src, trg)$ is the only domain attribute used in the chronicle models discussed in the previous section.

In [1], we argue that current alarm correlation approaches do not take advantage of all the available information, especially environmental information. For example, false alarms are often abusively attributed to poor intrusion detection

systems techniques : in many cases, false alarms are caused by the environment properties not being taken into account.

To address this, we proposed a formal data model called M2D2. M2D2 federates the information that is required for alarm correlation. In this section, we sketch how the chronicle recognition system can cooperate with the M2D2 framework.

4.1 Overview of M2D2

M2D2 can be seen as an infrastructure upon which alarm correlation systems can rely for events and structural information. M2D2 provides concepts and relations modeled with standard propositions of the classical first-order logic.

The concepts of M2D2 can be categorized in four groups : i) characteristics of the information system, ii) vulnerabilities, iii) security tools and iv) events.

Characteristics of the information system include information about the cartography[7] and the security policy. Vulnerabilities include information about the characteristics of known vulnerabilities : prerequisites, effect, products affected. Security tools include information about the nature and the configuration of the tools used to monitor entities of the information system for signs of attacks. Events include basic events (signs of the attacks) and alarms provided by IDSes, but also by the correlation systems.

A more detailed description of M2D2 can be found in [1].

4.2 M2D2 and the Chronicles

There is a two-way relationship between M2D2 and the chronicles : as a correlation system, the chronicles take advantage of the data provided by M2D2 and also act as an alarm provider for M2D2.

M2D2 data are modeled with first-order logic predicates, so they can be used as the atemporal information of the reified logic on which the chronicles are based. In other words, the M2D2's concepts are used as domain attributes of the chronicles. In our current approach, only M2D2's alarm concepts are used as domain attributes of chronicles (see section 3). However, the other available concepts would enhance the chronicle models.

The `infected` domain attribute in the example 3.4 is an example of the use of a domain attribute corresponding to M2D2's cartography class of information. We illustrate the use of topological information with another example : video conferences require the firewall to be open. Every participant is notified of the port number he/she shall use, and connections bursts on the server are observed. Thus, over these periods, portscans shall be described as false positives. By tracking the topological modification (the firewall opening), a chronicle could qualify the portscans as false positives during video-conference sessions.

[7] cartography both refers to the topology of the network and the softwares running on the hosts

In M2D2, the high level alarms triggered by correlation systems are related to lower level alarms with the *part_of* relation. Thus, alarms are structured in a hierarchy, where leaf nodes are the lowest level alarms provided by basic IDSes and root nodes are the highest level alarms, built with intermediate events. Only the hierarchies root alarms are directly shown to the operator. If detailed information about the alarms is required, the operator can browse the alarm hierarchy. There is a straightforward mapping between the IDWG [15] alarm structures and the event concept of M2D2.

Using previously recognized chronicles into other chronicle models is a functionality already included in the chronicle recognition system (see the looping functionality in section 2.1). By making M2D2 and the chronicle recognition system cooperate, the recognized chronicles are transformed into M2D2 high level alarms linked with the events with the *part_of* relation. The new alarms provided by the chronicle recognition system are *de facto* made available to other correlation systems relying on M2D2. This is illustrated in Figure 9. There are two alarm generators, a Snort sensor and CRS. When a nimda attack occurs, Snort generates the three `iis-decode-bug`, `iis-cmd-exe` and `iis-unicode` alarms. These alarms are made available to CRS, which recognizes a nimda attack. A `nimda` alarm is triggered, and related to the previous alarms with the *part_of* relationship. Only the nimda alarm is provided to the operator. On the contrary, the `shellcode` alarm is not related to any recognized chronicle so it is directly provided to the operator. The `nimda` alarm could be involved in another alarm via the *part_of* relation. For instance, we can use the vulnerabilities information contained in M2D2 to check whether the target server is really vulnerable or not (see [1]).

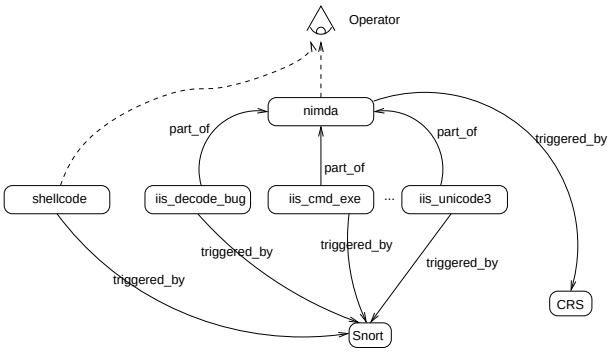

Fig. 9. Relations between alarms

The architecture of the global system is summarized in Figure 10. On this figure, we see that alarms provided by sensors (S_1 and S_2) are sent to a dispatcher. The dispatcher assigns unique identifiers to events and dispatches them

to the M2D2 database and to the correlation systems performing asynchronous (i.e. event-driven) analysis. Chronicle recognition is one of these asynchronous processes. Other environmental information contained in M2D2 are exploited as domain attributes of the chronicles.

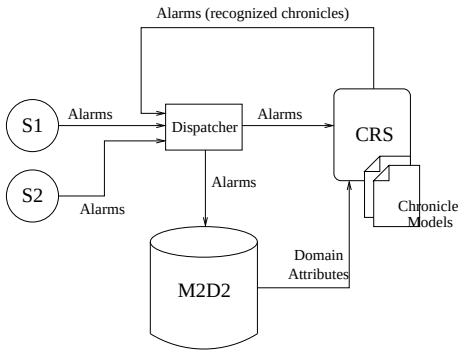

Fig. 10. Interactions between the chronicles and M2D2

5 Related Work

Among the six categories of languages proposed by Eckmann *et al* [2], two are of relevance here : detection languages and correlation languages. Our approach aims at correlating alarms, so we shall first compare to other correlation languages.

Cuppens [23] proposes a language, called Lambda, whose scenarios steps represent the attacker's action (be they observable or not). However, we believe that attackers strategies are too random to be the subject of explicit attack scenarios. In [24], Cuppens also proposes more flexible approaches to correlate alarms through the underlying attacks consequences and prerequisites.

Although chronicles could be used for this purpose, we do not dedicate chronicles to attack scenario modeling. In our approach, chronicles are used to represent known phenomena which involve several alarms and to strengthen and enhance single alarms by combining them with other events, as well as other information not found in the alarms.

The Statl [2] correlation language is a transition-based language, which is strongly dedicated to the underlying search algorithm, whereas the chronicles language is a high level declarative language. This language enabled to successfully apply chronicles to many distinct fields. Our intrusion alarm correlation component benefits from the operational and stable chronicle recognition system developed by Dousson for these application fields.

Another essential difference with the two previous works is that time is fundamental in chronicles, whereas the reasoning of Lambda and Statl is not based on time.

Although we are interested in correlating alarms, we shall compare our work with existing *detection* languages. Detection languages analyze raw events, some of which are manifestations of an attack. This is the fundamental difference between correlation and detection languages : in detection, the goal is to identify the events in the monitored stream that *are* suspects, among all the legitimate ones. In correlation, all events are potentially suspects.

However, from the point of view of the operators used in such languages, the distinction between detection and correlation languages is rather small. Thus, operators used in high-level declarative detection languages like Sutekh from Pouzol [21], LogWeaver from Goubault-Larrecq [3] are closer from our work than previously cited correlation languages. These languages could most likely be transposed to correlation languages (i.e take alerts as input instead of raw events).

Pouzol and Goubault-Larreq use non-reified temporal approaches; they take as input a trail, i.e a totally ordered set of events. From the point of view of expressiveness, since reified logics accord a special status to time and allow one to predicate and quantify over propositional terms, they are more expressive for classifying different types of temporal occurrence and in representing both non-temporal and temporal aspects of causal relationships.

As much as we know, Pouzol does not provide the counting predicate, but the chronicles would benefit of his work concerning the problem of the recognition being exhaustive, evoked in section 2.2 [22].

We way also compare our work with an existing alarm correlation system, called Risk Manager [8]. Risk Manager uses time information to aggregate alarms but is not *based* on a temporal reasoning. Moreover, it does not provide any mean to express explicit alarm sequences.

6 Conclusion and future work

In this paper, we proposed to apply chronicles to alarm correlation in intrusion detection. Chronicles benefit of strong theoretical background. They provide a high level declarative language which does not presume the nature of the underlying input events. An operational and stable implementation of the recognition system exists. Chronicles are indeed being successfully used in many distinct areas to monitor dynamic systems where the time information is relevant.

We have illustrated how chronicles might solve some of current intrusion detection issues like alarm overload, false positives and poor alarm semantics.

The proposed chronicles currently only use alarms as domain attributes. We plan to integrate the chronicle recognition system with an alarm correlation infrastructure, M2D2, in order to extend domain attributes to other relevant concepts, like topology, which is more and more dynamic.

Chronicle models are currently written by experts of the domain. A chronicle learning tool called *Face* is currently being developed by Dousson to discover frequent chronicles. In intrusion detection, many alarm groups are caused by recurrent phenomena, especially worms. We plan to apply the chronicle learning tool to discover such phenomena.

7 Acknowledgments

We are very greatful to Christophe Dousson for his help to use the Chronicle Recognition System. We would also want to thank Ludovic Mé and Mireille Ducassé for their comments.

References

1. B. Morin, L. Mé, H. Debar and M. Ducassé, "M2D2 : a formal data model for intrusion alarm correlation", *Proceedings of the 5th Recent Advances in Intrusion Detection 2002 (RAID2002)*, 2002.
2. S.T. Eckmann, G. Vigna, R.A. Kemmerer, "STATL : An Attack Language for State-based Intrusion Detection", *Dept. of Computer Science, University of California, Santa Barbara*, 2000.
3. M. Roger, J. Goubault-Larrecq, "Log Auditing Through Model-Checking", *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW01)*, 2001.
4. U. Lindqvist, P.A. Porras, "Detecting Computer and Network Misuse Through the Production-Based Expert System Toolset (P-BEST)", *Proceedings of the IEEE Symposium on Security and Privacy*, 1999.
5. D.V. McDermott, "A Temporal Logic for Reasoning about Processes and Plans", *Cognitive Science*, pp.101–155, 1982.
6. F. Bacchus, J. Tenenberg, J.A. Koomen, "A non-reified Temporal Logic", *Artificial Intelligence*, pp.87–108, 1991.
7. J. Allen, "Towards a General Theory of Action and Time", *Artificial Intelligence*, pp.123–154, 1984.
8. H. Debar, A. Wespi, "Aggregation and Correlation of Intrusion Detection Alerts", *Proceedings of the 4th Recent Advances in Intrusion Detection (RAID2001)*, October 2000.
9. S. Manganaris, M. Christensen, D. Zerkle, K. Hermiz, "A Data Mining Analysis of RTID Alarms", *Computer Networks: The International Journal of Computer and Telecommunications Networking*, Volume 34, Issue 34, October 2000.
10. C. Dousson, P. Gaborit, and M. Ghallab, "Situation Recognition : Representation and Algorithms", *in proceedings of the 13th IJCAI*, pp.166–172, August 1993.
11. C. Dousson, "Suivi d'évolutions et reconnaissance de chroniques", *PhD Thesis*, `http://dli.rd.francetelecom.fr/abc/diagnostic/`, 1994.
12. C. Dousson, "Alarm Driven Supervision for Telecommunication Networks : Online Chronicle Recognition", *Annales des Telecommunications*, pp.501–508, 1996.
13. C. Dousson, "Extending and Unifying Chronicles Representation with Event Counters", *in proceedings of the 15th European Conference on Artificial Intelligence (ECAI 2002)*, August 2002.

14. M. O. Cordier, C. Dousson, "Alarm Driven Monitoring Based on Chronicles", *in proceedings of the 4th Symposium on Fault Detection Supervision and Safety for Technical Processes (Safeprocess 2000)*, pp. 286–291, June 2000.
15. H. Debar, M.Y. Huang, D.J. Donahoo, "Intrusion Detection Exchange Format Data Model", *IETF Draft*, 2002.
16. Y. Shoham, "Temporal Logics in AI : Semantical and Ontological Considerations", *Journal of Artificial Intelligence*, pp.89–104, 1987.
17. R. Dechter, I. Meiri, J. Pearl, "Temporal Constraint Networks", *Artificial Intelligence*, pp.61–95, 1991.
18. G. Jakobson and M. D. Weissman, "Alarm correlation", *IEEE Network Magazine*, pp. 52–60, 1993.
19. K. Julisch, "Mining Alarm Clusters to Improve Alarm Handling Efficiency", *Proceedings of the 17th ACSAC*, December 2001.
20. S. Manganaris, *et al*, "A Data Mining Analysis of RTID Alarms", *First International Workshop on the Recent Advances in Intrusion Detection (RAID98)*, September 1998.
21. J.P. Pouzol, M. Ducassé, "From Declarative Signatures to Misuse IDS", *Proceedings of the 4th Recent Advances in Intrusion Detection (RAID)*, 2001.
22. J.P. Pouzol, M. Ducassé, "Formal Specification of Intrusion Signatures and Detection Rules", *Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW)*, 2002.
23. F. Cuppens, "Managing Alerts in Multi-Intrusion Detection Environment", *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC 01)*, 2001.
24. F. Cuppens, A. Miege, "Alert Correlation in a Cooperative Intrusion Detection Framework", *Proceedings of the IEEE Symposium on Security and Privacy"*, 2002.