A Scalable and Efficient Correlation Engine to Detect Multi-step Attacks in Distributed Systems

David Lanoe, Michel Hurfin, Eric Totel

A Scalable and Efficient Correlation Engine to Detect Multi-step Attacks in Distributed Systems

David Lanoe
CentraleSupelec, INRIA, IRISA
Rennes, France,
David.Lanoe@centralesupelec.fr

Michel Hurfin
INRIA, Univ Rennes, IRISA
Rennes, France
Michel.Hurfin@inria.fr

Eric Totel
CentraleSupelec, INRIA, IRISA
Rennes, France
Eric.Totel@centralesupelec.fr

***Abstract*—In distributed systems and in particular in industrial SCADA environments, alert correlation systems are necessary to identify complex multi-step attacks within the huge amount of alerts and events. In this paper we describe an automata-based correlation engine developed in the context of a European project where the main stakeholder was an energy distribution company. The behavior of the engine is extended to fit new requirements. In the proposed solution, a fully automated process generates thousands of correlation rules. Despite this major scalability challenge, the designed correlation engine exhibits good performances. Expected rates of incoming low level alerts approaching several hundreds of elements per second are tolerated. Moreover, the used data structures allow to quickly handle dynamic changes of the set of correlation rules. As some attack steps are not observed, the correlation engine can be tuned to raise an alert when all the attack steps except k of them have been detected. To be able to react to an ongoing attack by taking countermeasures, alerts must also be raised as soon as a significant prefix of an attack scenario is recognized. Fulfilling these additional requirements leads to increase the memory consumption. Therefore purge mechanisms are also proposed and analyzed. An evaluation of the tool is conducted in the context of a SCADA environment.**

I. Introduction

Many domains (industry, finance, health, education, culture, housing, transportation, ...) are now managed by information and communication technology (ICT) systems. All these ICT systems represent potential targets for malicious attackers that aim at compromising their availability, integrity or confidentiality properties. Designing and implementing secure software and hardware systems is an ideal goal but it is rarely fully attained during the software conception phase. As the ICT systems are continuously growing in size and complexity, critical vulnerabilities remain in many systems and can be exploited by attackers as long as they are not patched. Thus, in addition to prevention approaches, supervision mechanisms are needed to detect on-line attacks. Once an alert is raised, appropriate countermeasures may be considered to reduce the impact of the suspected attack. A supervision process relies on intrusion detection systems (IDSes) installed at different places of an ICT system (both in the network and on the hosts) and configured to monitor local actions and to report occurrences of event (normal actions) as well as raw alerts (detected anomalies).

The numerous deployed IDSes produce a huge number of alerts. IDSes are often inaccurate and a large number of these alerts are false positives that overwhelm the administrator of a system. The original purpose of an alert correlation process was thus (1) to reduce the number of false positives and (2) to propose high semantic alerts to the administrator when an actual attack occurs. In particular, a correlation engine should be able to consider multi-step attacks and to warn the administrator when such a complex and meaningful attack occurs. In order to detect the chained actions of an attacker, the correlation component needs to rely on a set of correlation rules: each rule describes a known multi-step attack that may affect the system. The creation of these correlation rules is not trivial. Thus in current industrial SIEM (Security Information and Event Management) products, the defined rules are still either poor or incomplete.

The work we present in this paper has been conducted in the context of the PANOPTESEC European project. The purpose of this project, which ended in 2017, is to produce a supervision platform for a SCADA system managed by a major electricity distribution company in Italy. Due to the criticality of the industrial context, additional requirements have to be satisfied by the alert correlation process which is at the core of the platform. First, as it is difficult to write correlation rules, they are produced by a fully automated process. As a consequence, their number is much higher. Moreover, this set of several thousands rules has to be updated when the system configuration evolves. Consequently, while handling often hundreds of alerts per second, the correlation engine must also quickly reconfigure itself to ignore the suppressed rules and to cope with the new created ones. Second, the administrator wants to be aware of the actual attacks in real time while they are under progress. To be able to early apply countermeasures, the correlation engine should produce an alert before all the steps of an attack have been completed. Third, at a given step of the attack, an attacker can evade detection due to failures, misconfigurations or limitations of some IDSes. Due to these false negatives, one or several attack steps do not appear in the flow of processed alerts. Consequently, the correlation engine must be able to detect multi-step attacks despite a few missing steps.

The contribution of our work, with respect to the state of the art, is to design, implement and assess a correlation engine called ABE (Automaton Based Engine) that is able:

- to take as input automatically generated correlation rules;

- to predict incoming multi-step attacks;
- to detect attack scenarios with some missing steps;
- to scale to handle hundreds of alerts per second while supporting thousands of correlation rules.

Over the past fifteen years, works [1] have followed an automata based approach to develop correlation engines. More recently, the problem of automated generation of correlation rules has been studied in [2]. However scalability issues raised by the mixing of both objectives have not been addressed. Furthermore to promote the use of such solutions, the requirements of the administrators have to be taken into account and practical problems related to memory consumption should not be ignored.

This paper is organized as follows. The next section discusses related work. The requirements the correlation engine has to fulfill are detailed in Section 3. Section 4 presents our correlation engine. An evaluation is proposed in Section 5 and Section 6 concludes the paper.

II. State of the Art: Alert Correlation

An alert correlation process is composed of several phases [3]. The first one consists in normalizing the format of heterogenous events and alerts. After this homogenization phase, the diversity of the data sources is no more a problem. The second phase consists in verifying that an alert is correct (or at least it seems plausible). A base of knowledge describing the supervised system (the topology, the cartography, the base of vulnerabilities) is often used to verify if an alert is correct or not [4]. False positives are suppressed during this verification phase (e.g., an alert reveals an attack step against a process that does not exist). The third phase consists in merging alerts that are produced by various IDSes reacting to the same attack. This merge is also called *Alert Fusion*. Within this paper, once the third phase is achieved, the remaining events and alerts are called low level alerts. In a fourth and last phase, the flow of low level alerts is processed by a correlation engine to discover possible occurrences of complex attack scenarios.

Many correlation engines rely on a base of correlation rules which identifies all the patterns to look for in the flow of low level alerts. Several langages have been previously defined to describe correlation rules such as ADeLe [5], LAMBDA [6] or STATL [7]. An attack scenario is characterized by several attack steps against the system and some temporal relationships between them. A correlation rule describes an attack scenario and indicates, for each attack step, how to find within the incoming flow of low level alerts, an element that attests that this step has been performed. Indeed, one of the difficulties is to connect a low level alert (i.e., an observation performed by an IDS) and an attack step (i.e., an action performed during the computation). If low level alerts produced by the deployed IDSes match with the description of a correlation rule, an alert has to be raised by the correlation engine.

Most of the SIEMs on the market (for example, IBM QRadar or HP Arcsight) include an alert correlation mechanism configured by alert correlation rules. Orchids [8], GNG [1] or Hi-DRA [9] are well-known examples of more sophisticated academic correlation engines that use also correlation rules. In all these works, the production of the correlation rules requires to precisely know the attacker's strategy and how the IDSes may (or may not) detect his activities. When it is done by human experts, this is a non trivial, time consuming, and error-prone task. Thus a recent work [2] aims at producing the correlation rules through an automated process.

Obviously, to produce a correlation rule, it is necessary to know how an attacker performs progressively his attack against the system. The representation of the attacker's activities can be made in different ways. The use of attack trees [10] is an usual approach. In these trees, the nodes are logical nodes (operators AND and OR), and the leafs are the attacker's actions. This representation of the attacker's behavior can be transformed into a correlation tree that represents all the sequences of events or alerts an attacker can generate in the supervised system. This approach has been followed in [2] and demonstrates that an attack can be associated with two similar representations corresponding respectively to the point of view of the attacker (attack tree) and the point of view of the defender (correlation tree). A base of knowledge about the system can be used to develop an automated process which establish the links between actions and possible observations and thus deduce the correlation rules from the attacker description [2]. Of course this requires to describe the attacker actions. Indeed there are well known research works such as MulVal [11] that generates automatically the attacker behavior in a distributed environment: this work consists in generating attack graphs. There are various types of attack graphs. MulVal generates logical attack graphs where the nodes are states of the system, and the vertices are the attacker actions. Another sort of attack graph is called topological attack graph [12]. In these graphs, each node is a machine in the system, and each vertex the exploitation of a vulnerability that permits for an attacker to gain access from a source machine to a targeted machine.

The work we present here is a follow-up of the correlation based on a set of rules. However, the form of the correlation rules, the way they are produced and used are slightly different. Moreover, compared to various approaches, we scale to a large number of alerts per second and correlation rules.

III. The context of the PANOPTESEC project

Information about all the aspects of the PANOPTESEC project can be found in the dedicated web site [13]. In this section, we give hints about the components in charge of providing inputs to the correlation engine (the flow of low level alerts and the set of correlation rules). Then the specific requirements that have to be fulfilled by the correlation engine are detailed.

A. Correlation Rules Generation

In the PANOPTESEC project, scanners of vulnerabilities and topology discovery techniques are used to generate topological attack graphs. In a first step, information about the devices present in the system, their known vulnerabilities

and their connections are used to determine all the paths in the supervised system that can be followed by an attacker to reach and compromise some identified targets. A simple solution may consist in enumerating all attack paths in order to generate attack rules as sequences of attacker actions. However, this approach clearly does not scale. In the proposed approach, the generated graph is used to compute spanning attack trees but only scenarios that do not exceed 4 to 5 steps are considered. In practice, an attacker prefers to follow a short path. Furthermore a longer path often contains all the attack steps of a shortest one already identified. We suppose also that the attacker is coming from outside the supervised system. These realistic assumptions significantly reduce the number of attack paths. The trees obtained are in fact attack trees where each attacker's action corresponds to a vulnerability exploited to perform the attack step. Through the configuration of the deployed IDSes and thanks to the first steps of the correlation process (normalization, enrichment, fusion), the identification of the vulnerability (i.e., a CVE identifier) is also contained in most low level alerts that are generated when a vulnerability is exploited. Informally, a low level alert matches with an attack step of an attack path if 1) the two mentioned vulnerabilities are similar and 2) the identification of the source and target devices involved in this exploit are also the same.

Each correlation rule is represented as a correlation tree. Figure 1 is an example of a correlation tree where only two operators are used. The SEQ operator (sequence) states that all the children must be detected from left to right. The AND operator requires that all the children must be detected in any order. A tree may also use an operator OR to indicate that at least one of the children must be detected. In this example, the rule defines that the attack steps A, B, C and D must be detected in at least one of the orders $\{$A, B, C, D$\}$ or $\{$A, C, B, D$\}$, leading to the definition of two sequences of alerts. Note that uppercase letters (A, B, C, ...) are used to identify the steps of an attack described in a rule (i.e., types of alerts) while lower case letters (a, b, c, ...) refer to low level alerts. We use the same letter to indicate that a low level alert b matches with an attack step B. In the incoming flow, the different occurrences of low level alerts that match with an attack step B are denoted b_1, b_2 and so on.

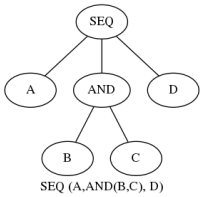

Fig. 1: A Correlation Tree

The correlation engine analyzes the incoming flow of low level alerts. Assume that this flow begins with the prefix "z_1; a_1; d_1; c_1; b_1; d_2; c_2; d_3". At this stage, the correlation engine should have raised three alerts because the sub-sequences "a_1; c_1; b_1; d_2" , "a_1; c_1; b_1; d_3", and "a_1; b_1; c_2; d_3" are corresponding to three different attacks that occur concurrently and match the attack scenario depicted by the above rule.

The generation of correlation rules is done at the very beginning of the computation. Yet the configuration of the system evolves. Changes in the topology and connectivity may have various legitimate causes: in particular, they may be the result of countermeasures applied when ongoing attacks are detected. In this work, the set of correlation rules used by the correlation engine can be updated at any time (See Section IV-E).

B. Specific Requirements

As indicated before, the requirements identified during the project were slightly different from those usually adopted for a standard correlation engine.

1) Ongoing Alerts: First of all, an engine usually produces an alert only after an attack scenario is entirely detected. In the previous example, we supposed that this usual strategy was adopted. Providing information about the progress of an attacker was a major requirement in the context of the project. Indeed, identifying a prefix of an attack scenario allows to identify the possible targets of an attacker and to consider quickly possible countermeasures. A visualization component has been developed to give a simple and rich view of the attacker's actions while the actual attack progresses. Even if a scenario is not finished, an early information may allow the administrator to predict and understand the possible goals of an attacker while being aware of the risk of error in the prediction. When the correlation engine raises an alert before the detection of the full scenario, the alert is called an *Ongoing Alert*. Consider again the previous example. At most five Ongoing Alerts can be generated corresponding respectively to the prefixes "a_1;", "a_1; c_1", "a_1; b_1","a_1; c_1; b_1", and "a_1; b_1; c_2". To limit the number of raised alerts and to reduce the risk of error in the prediction, Ongoing Alerts corresponding to the shortest prefixes should not be raised. In practice, for each rule, we consider the length of the longest attack scenario and the length of the prefix already observed. A threshold is defined to control the amount of generated Ongoing Alerts. For example, for a sequence of 5 steps, if the threshold is set to 40%, an alert is generated if the observed prefix identifies already more than 2 attack steps.

2) Missing Alerts: Secondly, we must consider that an attacker can evade IDSes detection. For example, it could be the case that no IDS has the right location and the right specification/configuration to detect the occurence a_1. It could be the case also that an IDS fails to detect an attack step or the information it provides in the low level alert is not rich enough to discover that a_1 matches with the attack step A. In an usual correlation engine, this means that the scenario will not be detected. In this work, the correlation engine is designed to detect scenarios with missing steps and to generate a *Missing Alert* in that case. Obviously only a few missing low level alerts can be tolerated. Otherwise the accuracy of the detection is too low. An upper bound Max_k is defined: this bound must depend on the length of the longest scenario

defined by a correlation rule. When the length of the attack paths is rather small, the value of Max_k is set either to 0 or 1.

3) Most Advanced Detection: The two above requirements may cause a significant increase both in the number of raised alerts and in the amount of memory space used by the correlation engine. An analysis of the procedures followed by the stakeholder indicates that, for a given attack path, the administrator is mainly interested by the level of progression of the attack(s). Moreover, when several low level alerts match with the same attack step of an attack path, the administrator focuses much more on the most recent one. Based on the fact that, at any time, for each attack path, the last most completed scenario has more interest, an alternative behavior for the correlation engine can be defined to focus on the progress of an attack. In the context of this project, the automated generation of attack rules ensures that the root of a correlation tree is always a SEQ operator. Consequently, this representation already identifies some major steps in the attack scenario that have to be achieved sequentially by an attacker. Thus the concept of *most advanced detection* (See Section IV-D) is natural and straightforward.

4) Performances: Throughout the design of the correlation engine, performance is a critical aspect. The main challenge is to obtain a solution able to cope with thousands of rules and hundreds of low level alerts per second. As the set of correlation rules is updated from time to time, the time required to initialize the data structures of the correlation engine is also critical. Indeed, while it is in the process of resetting, the analysis of the flow of low level alerts is suspended. New low level alerts that arrive are stored. During the reset time, the low level alerts that are waiting for a treatment are accumulated. When the correlation engine resumes its analysis, it must catch up with the accumulated delay.

IV. Automaton Based Engine (ABE) Description

A. Automaton Generation

As indicated previously, a correlation rule is represented by a correlation tree. However, using these trees directly at detection time is not the best choice in term of performance. The goal is to raise alerts. Each alert must identify a particular attack scenario and a sub-sequence in the flow of low level alerts such that 1) each element of this sub-sequence matches with an attack step of the scenario (and vice versa) and 2) the temporal constraints between the attack steps are also respected by the sub-sequence. Low level alerts and attack steps can be attached to letters of an alphabet; the sub-sequence is a word and the rule is a language (i.e., a set of words). To detect and attest that a sub-sequence is an attack, a natural choice is to associate each rule with an automaton. Each time a new set of correlation rules is computed, the correlation trees are loaded and each of them is transformed into an automaton that is able to recognise the sequences of alerts specified by the correlation tree.

Figure 2 shows the automata that are generated in three particular cases where the correlation tree contains a single

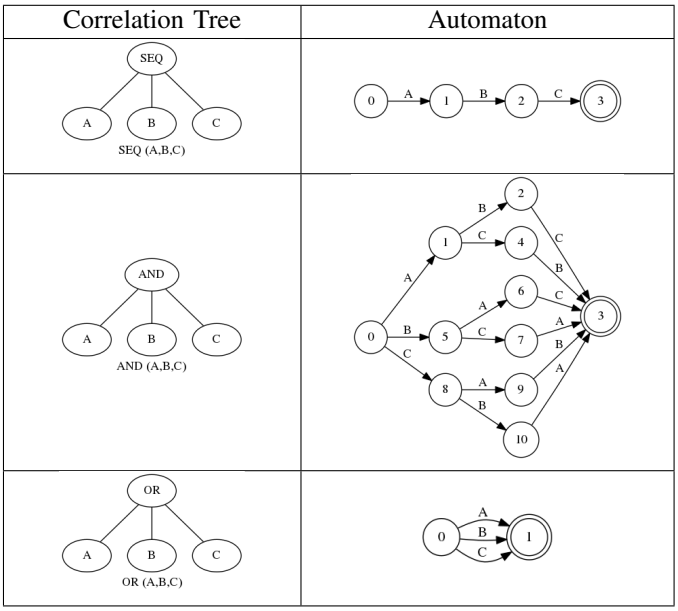

Fig. 2: Basic Correlation Trees and their Automata

logical operator. For example, a sequence of three alerts A, B and C generates an automaton that recognises only the sequence A, B, C. The AND operator for three alerts A, B, and C accepts all sequences containing the three alerts in any order. Finally the OR operator between three alerts A, B, and C is corresponding to an automaton that accepts at least one of these alerts.

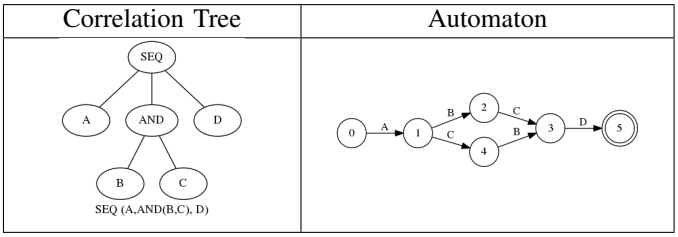

Fig. 3: A More Complex Correlation Tree and its Automaton

When a correlation tree is a mix of these three operators, the whole automaton is the result of the composition of the automata associated to each internal node. Figure 3 shows the correlation tree already presented in Figure 1 and its automaton that accepts the two sequences of alerts defined by the correlation rule. This automaton is the composition of the two automata generated for each used operators (i.e., SEQ and AND nodes).

B. Engine Attack Recognition Algorithm

Without loss of generality, we consider in a first step that the correlation engine is fed with a single correlation rule associated to a particular automaton $\mathcal{A}$. Let us describe first a simple and inefficient solution: the correlation engine enumerates all the possible sub-sequences contained in the

flow of low level alerts and tests for each of them if the corresponding word is accepted or not by the automaton. As the flow of low level alerts contains a lot of elements that are of no interest for the detection, creating the list of possible sub-sequences will be a time consuming operation and will require to keep track of the whole flow of low level alerts. Moreover, all the sub-sequences that share a non-matching prefix will be systematically enumerated and rejected.

The proposed solution follows a different strategy. We consider prefixes of sub-sequences. At any time, we keep track of all the prefixes that have been already observed and not rejected by the automaton. After the initialization phase, the empty word is the only prefix already observed and not rejected. During the analysis, the element of the incoming flow are taken into account sequentially, one by one. For each low level alert e, the correlation engine checks if an existing prefix $\mathcal{H}1$ can be extended with this new letter to obtain a new prefix $\mathcal{H}2$ (equal to $\mathcal{H}1 + e$) that is not rejected by the automaton. When this is the case, the new prefix is saved. For efficiency reason, the state of the automaton after the prefix recognition is also saved: thus rather than checking entirely a new prefix, only the existence of a transition labelled with the new low level alert e is verified. The saved information is structured as a collection of *Plans*. A plan $\mathcal{P}$ is a particular instance of the execution of an automaton $\mathcal{A} : \mathcal{P} = \{\mathcal{A}, \mathcal{S}, \mathcal{H}\}$ where S is the state reached during this particular execution of the automaton A and $\mathcal{H}$ is the history of low level alerts that have been recognized to reach this state S (i.e., $\mathcal{H}$ is a prefix). When S is not a final state, $\mathcal{H}$ is a sequence of low level alerts corresponding to the prefix of a known sequence of attack steps depicted by the automaton. If S is the final state of an automaton, then $\mathcal{H}$ is the sequence of alerts corresponding to a complete multi-step attack.

Assuming that $Max_k = 0$ (i.e., no missing alert is raised), the Algorithm 1 defines how the plans are managed by the correlation engine. In the pseudo code, the word event is used to refer to a low level alert. In practice, each plan subscribes to types of events that can permit to advance in the recognition (line 29). When a low level alert arrives (line 8), we build the set of plans that are concerned with this event (line 12). Then, for each of these plans, we create a new plan that is in the new state reached after firing the transition associated to the low level alert (lines 14-17). After this operation, the plan subscribes to the events that it is waiting for (line 23). If the plan created is in its final state, then an Alert (called a complete and exact alert) is generated, and the created plan disappears (lines 18-20). If it is not in a final state, it is added to the set of plans (line 30). In this case, an Ongoing Alert is generated if the prefix already recognized is longer than a threshold defined for each automaton. Note that, in this algorithm, the matching between a low level alert and an attack step is checked at line 12.

Lets illustrate this algorithm on the example of Figure 3. We suppose that the correlation engine treats the events a_1; d_1; b_1; e_1; c_1; d_2 that are of types A, D, B , E, C and D. Table I describes the different computation steps. It must be read from first line to last line. At the initialisation, for each automaton, a plan that is in the initial state (0) and has an empty recognition history is created. This plan also subscribes to all event types that can fire a transition from the initial state. Here p_0 subscribes to event type A. When a_1 is generated and received, the correlation engine identifies the set of plans that have subscribed to events of type A. In our case, this set is composed of plan p_0. A new plan p_1 is created on the basis of p_0: the plan p_1 fires the transition A from state 0 of p_0 and reaches state 1. Its history also extends to event a_1, that contributed to the recognition of a new attack step. The plan p_0 remains in memory, as it can detect new attack beginnings. The process is repeated for each alert. In this example, when d_2 is released, we generate a plan that reaches the final state (5), with an history $\{a_1, b_1, c_1, d_2\}$. This history represents

Algorithm 1 Plan management algorithm

```
1: procedure INITPLANS(A)
2:     p ← NEWPLAN
3:     p.A ← A
4:     p.H ← ∅
5:     p.S ← 0
6:     COMPUTEFUTURE(p)
7: end procedure
8: procedure RECOGNISEATTACK(event)
9:     MANAGEPLANS(event)
10: end procedure
11: procedure MANAGEPLANS(event)
12:     P ← PLANSSUBSCRIBED(event)
13:     for all P_i in P do
14:         p ← NEWPLAN
15:         p.A ← P_i.A
16:         p.S ← NEXTAUTOMATONSTATE(p.A, P_i.S, event)
17:         p.H ← P_i.H+event
18:         if ISFINALSTATE(p.S) then
19:             GENERATEALERT(p)
20:             DELETEPLAN(p)
21:         else
22:             GENERATEONGOINGALERT(p)
23:             COMPUTEFUTURE(p)
24:         end if
25:     end for
26: end procedure
27: procedure COMPUTEFUTURE(p)
28:     future ← NEXTTRANSITIONS(p.A, p.S)
29:     SUBSCRIBE(p, future)
30:     P ← P ∪ {p}
31: end procedure
```

TABLE I: Recognition process

Low Level Alerts	Parent Plan	New Plan	Subscribed Event
-	-	p_0 = {$\mathcal{A}$, (0), { }}	A
a_1	p_0	p_1 = {$\mathcal{A}$, (1), {a_1}}	B, C
d_1	-	-	-
b_1	p_1	p_2 = {$\mathcal{A}$, (2), {a_1, b_1}}	C
e_1	-	-	-
c_1	p_1	p_3 = {$\mathcal{A}$, (4), {a_1, c_1}}	B
	p_2	p_4 = {$\mathcal{A}$, (3), {a_1, b_1, c_1}}	D
d_2	p_3	p_5= {$\mathcal{A}$, (5), {a_1, b_1, c_1, d_2}}	**Complete & Exact Alert**

the attack steps as seen by the supervision mechanisms, and matches the correlation rule of Figure 3. Note that the last plan appears in a smaller font size to underline the fact that this plan is stored temporarily (line 20).

Let us now consider that several correlation rules exist. Each plan is associated to a single automaton. Yet plans related to different automata can be mixed in some data structures. For example, the set P that contains all the plans that have subscribed to a same event (line 12) can refer to distinct correlation rules. The fact that the correlation rules are not managed separately explains the good performance of the algorithm.

The above algorithm lacks one of the requirements: if some low level alerts have not been generated or if the information it carries does not allow to detect a matching with the corresponding attack step, the attack remains undetected. The algorithm will now be enhanced to be able to tolerate missing alerts.

C. Managing Missing Low Level Alerts

We designed an adaptation of Algorithm 1 in order to tolerate up to Max_k missing alerts in the sub-sequences expected by an automaton. Algorithm 2 is called the k-missing algorithm. Note that when $Max_k = 0$, Algorithm 2 behaves exactly like Algorithm 1. In this new algorithm, the definition of a plan $\mathcal{P}$ changes to $\mathcal{P} = \{\mathcal{A}, \mathcal{S}, \mathcal{H}, \text{k}\}$ where S and $\mathcal{H}$ have the same meanings as before, but an integer k is now attached to each plan. By construction, $0 \leq k \leq Max_k$. Roughly speaking, k is the number of remaining jokers that can be used during the next steps of the scenario recognition. Consequently, $Max_k - k$ indicates how many low level alerts are already missing in the history $\mathcal{H}$.

During the initialisation of the algorithm, the value of k for the first plan p_0 is set to Max_k. For sake of simplicity we assume that this initial bound is the same for all the correlation rules but, of course, it can be adjusted to fit with the complexity of each attack scenario (i.e., its number of attack steps). If the value of Max_k is not equal to 0, all the plans corresponding to possible prefixes of length 1 have to be created. These new initial plans have a value k equal to p_0.k-1. Then the initialization process continues for plans with a unobserved prefix of length 2, 3, ..., p_0.k. During the analysis, when a low level alert is received, the correlation engine calls *recogniseAttack*. This procedure allows to indicate that the event is not missing (line 10, call of the procedure *managePlans* with a boolean parameter equal to false). Like in Algorithm 1, new plans are built after firing the transitions associated to this low level alert (lines 15-18). Then, for each new plan p, new plans (where up to p.k low level alerts can be missing) are also built (line 39, call of the procedure *managePlans* with a boolean parameter equal to true).

To understand the k-missing algorithm, we consider again the example already used to explain Algorithm 1. As a reminder, the correlation rule and the corresponding automaton are depicted in Figure 3. We suppose that the observed flow of low level alerts is a_1; d_1; b_1; e_1; c_1; d_2. Note that, in this explanation, we consider exactly the same flow while the management of missing low level alerts is done to tolerate the lack or an alteration of some elements within the flow. Assuming that $Max_k = 1$, the created plans are shown in Table II. At initialisation, the first plan p_0 in state 0 with an empty history is created. As $Max_k = 1$, we compute also plans as if the events in the immediate future have occurred (but have not been observed). Only one plan p_1 is created (after the transition A). We include in its history the event type A as a missing event (noted $\overline{A}$). In this example, after the initialisation phase, two plans are waiting for different event types. When a_1 is received, it can only concern p_0. We then generate p_2 (with p_2.k=p_0.k), and the plans in a future of length 1, that are p_3 and p_4. The algorithm goes on until we consume all received alerts. At the end of the 1-missing algorithm, if we launch alerts only for completely recognized signatures, we generate 7 alerts, compared to 1 alert in the first algorithm. Moreover,

Algorithm 2 Plan management k-missing algorithm

```
 1: procedure INITPLANS(A, max_k)
 2:     p ← NEWPLAN
 3:     p.A ← A
 4:     p.H ← ∅
 5:     p.S ← 0
 6:     p.k ← max_k
 7:     COMPUTEFUTURE(p)
 8: end procedure
 9: procedure RECOGNISEATTACK(event)
10:     MANAGEPLANS(event, false)
11: end procedure
12: procedure MANAGEPLANS(event, missing)
13:     P ← PLANSSUBSCRIBED(event)
14:     for all P_i in P do
15:         p ← NEWPLAN
16:         p.A ← P_i.A
17:         p.S ← NEXTAUTOMATONSTATE(p.A, P_i.S, event)
18:         p.k ← P_i.k
19:         event.missing ← missing
20:         if missing then
21:             p.k ← p.k − 1
22:         end if
23:         p.H ← P_i.H+event
24:         if ISFINALSTATE(p.S) then
25:             GENERATEALERT(p)
26:             DELETEPLAN(p)
27:         else
28:             GENERATEONGOINGALERT(p)
29:             COMPUTEFUTURE(p)
30:         end if
31:     end for
32: end procedure
33: procedure COMPUTEFUTURE(p)
34:     future ← NEXTTRANSITIONS(p.A, p.S)
35:     SUBSCRIBE(p, future)
36:     P ← P ∪ {p}
37:     if p.k > 0 then
38:         for all event in future do
39:             MANAGEPLANS(event, true)
40:         end for
41:     end if
42: end procedure
```

TABLE II: Recognition process 1-missing Algorithm

Low Level Alerts	Parent Plan	New Plan	Subscribed Event
-	-	p_0 = {$\mathcal{A}$, (0), {}, k=1}	A
	p_0	p_1 = {$\mathcal{A}$, (1), {$\overline{A}$}, k=0}	B, C
a_1	p_0	p_2 = {$\mathcal{A}$, (1), {a_1}, k=1}	B, C
	p_2	p_3 = {$\mathcal{A}$, (2), {$a_1, \overline{C}$}, k=0}	B
	p_2	p_4 = {$\mathcal{A}$, (2), {$a_1, \overline{B}$}, k=0}	C
d_1	-	-	-
b_1	p_1	p_5 = {$\mathcal{A}$, (2),{$\overline{A}, b_1$}, k=0}	C
	p_2	p_6 = {$\mathcal{A}$, (2), {a_1, b_1}, k=1}	C
	p_3	p_7 = {$\mathcal{A}$, (2), {$a_1, \overline{C}, b_1$}, k=0}	D
	p_6	p_8 = {$\mathcal{A}$, (3), {$a_1, b_1, \overline{C}$}, k=0}	D
e_1	-	-	-
c_1	p_1	p_9 = {$\mathcal{A}$, (4), {$\overline{A}, c_1$}, k=0}	B
	p_2	p_{10} = {$\mathcal{A}$, (4), {a_1, c_1}, k=1}	B
	p_{10}	p_{11} = {$\mathcal{A}$, (3), {$a_1, c_1, \overline{B}$}, k=0}	D
	p_4	p_{12} = {$\mathcal{A}$, (3), {$a_1, \overline{B}, c_1$}, k=0}	D
	p_5	p_{13} = {$\mathcal{A}$, (3), {$\overline{A}, b_1, c_1$}, k=0}	D
	p_6	p_{14}= {$\mathcal{A}$, (3), {a_1, b_1, c_1}, k=1}	D
	p_{14}	p_{15}= {$\mathcal{A}$, (5), {$a_1, b_1, c_1, \overline{D}$}, k=0}	**Complete & Missing**
d_2	p_7	p_{16} = {$\mathcal{A}$, (5), {$a_1, \overline{C}, b_1, d_2$}, k=0}	**Complete & Missing**
	p_8	p_{17} = {$\mathcal{A}$, (5), {$a_1, b_1, \overline{C}, d_2$}, k=0}	**Complete & Missing**
	p_{11}	p_{18} = {$\mathcal{A}$, (5), {$a_1, c_1, \overline{B}, d_2$}, k=0}	**Complete & Missing**
	p_{12}	p_{19} = {$\mathcal{A}$, (5), {$a_1, \overline{B}, c_1, d_2$}, k=0}	**Complete & Missing**
	p_{13}	p_{20} = {$\mathcal{A}$, (5), {$\overline{A}, b_1, c_1, d_2$}, k=0}	**Complete & Missing**
	p_{14}	p_{21} = {$\mathcal{A}$, (5), {a_1, b_1, c_1, d_2}, k=1}	**Complete & Exact**

for the same automaton and the same alerts as inputs, we generate 5 plans in the first algorithm, compared to 21 plans in the 1-missing algorithm.

As a consequence, we can guess that the k-missing algorithm is a very memory consuming algorithm. The high production of alerts is a side-effect that is not very compliant with the objectives of alert correlation. Indeed, we produce so many alerts that the administrator will again be overwhelmed under the mass of alerts. However, in the context of the project, it permits to visualise in real-time potential attacks with missing attack steps. As it is visually treated, the administrator does not have to dig into logs. The visualisation is the key functionality that permits to treat such a volume of alerts in real-time.

Obviously, adopting high values for Max_k is not a good idea. As indicated before, for each rule, the initial value of Max_k can be adapted according to the length of the scenario. For example, when the scenario has a length less than 3, the mechanism to tolerate missing low level alerts can be inhibited (i.e., $Max_k = 0$). When the length is greater than 10, considering up to 2 missing alerts is a reasonable choice. Otherwise $Max_k = 1$ is a good tradeoff.

The k-missing algorithm can also be coupled with Ongoing Alerts. This means that if the history of a plan is higher than a given length, an Ongoing Alert can be generated. Of course, these mechanisms have to be introduced carefully, as they can generate a high number of intermediate alerts.

D. Memory Consumption

As seen in Table II, the use of the k-missing algorithm generates in memory a lot of plans, that represent ongoing attack recognitions. Even when Max_k is set to 0 (Algorithm 1), the number of plans continuously increases. As the intrusion detection process is an online everlasting process, the number of plans depends mainly on the number of low level alerts that match with attack steps. These plans should be kept in memory as long as necessary in order to detect further attack steps that could occur within hours or days. However, the accumulation of plans results rapidly in an exhaustion of the memory. This implies that we must carry out a garbage collector to reduce the memory consumption.

In practice, when the correlation engine heap is full, we must delete some plans that are not useful for future recognitions. As it is difficult to predict the future, we have to implement heuristics to decide which plans must be deleted. Without any additional knowledge, the plans that are least likely to be useful to recognize an attack are the oldest plans. This means, that we can arbitrarily decide to suppress from memory for example 20% of the oldest plans. However, this heuristic is not sufficient, as it can suppress almost recognized correlation rules. Even if it seems reasonable, the above strategy relies on the assumption that performing a complex attack is never spread over a long period of time. This hypothesis is very questionable.

To tackle this problem, we propose to tag plans that are of interest (and not necessarily the most recent ones). As indicated previously, in the context of this project, it has been stated that administrators are interested by the most advanced scenarios and among these scenarios by one of the most recent ones. As the correlation tree attached to a correlation rule is such that its root is always a SEQ operator, it is simple to identify a sequence of steps that have to be performed by the attacker. Thus we can observe a progress level in the recognition process. This notion of level is used to manage the threshold mechanism that controls the generation of the Ongoing Alerts. It can also be used to identify plans that allow to reach a particular level. For a given rule and for each level already reached, we select a particular plan called the most advanced plan. By construction, each new plan that allows to reach a level in the recognition process is a most advanced plan at the time of its creation. The plan may lost its privilege later if, in the future, a new low level alert allows to reach exactly the same level of recognition in the same scenario. During the purge performed by the garbage collector mechanism, any plan with this special tag is kept in memory. Thus we avoid to loose recognitions that may be of interest in the future even if the attack steps already performed are very old. Note that a purge mechanism may keep only the most advanced plans. In that case, the number of plans is bound. If r is the number of correlation rules and if l is the maximum number of levels in any rule, at most $r.l$ plans are kept in memory.

E. Reaction to Dynamic System Changes

Till now, the presentation of the algorithm assumes that the base of correlation rules is statically defined once and for all. This does not reflect the reality, where an equipment can be removed or inserted dynamically in the information system:

this modifies the topology of the network and the vulnerabilities present in the system. Moreover any countermeasure adopted by an administrator to block an attacker may also have an impact on the topology (a machine is stopped), the connectivity (a port is closed) or the present vulnerabilities (a patch is applied). Thus dynamic aspects of the supervised system must be taken into account.

In practice, the attack graphs are periodically computed on the system. If necessary an update of the correlation rules is launched. Once a modification on the set of rules is triggered (at time t_1), the correlation engine suspends its analysis of the flow of low level alerts to update its data structures. When a correlation rule is suppressed, the corresponding automaton and all the associated plans are removed. When an new rule is defined, the initial plans are created.

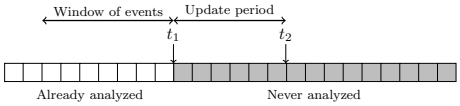

Fig. 4: Log of Events when an Update of the Set of Rules Occurs

When the analysis can start again (at time t_2, Figure 4), for all the rules, the correlation engine analyzes the flow of incoming low level alerts from the point where it stopped (time t_1). Yet, regarding the new rules (and only for them), an alternative solution consists in replaying part of the most recent low level alerts that have been already considered before the new set of rules has been adopted. When a change occurs, the time that elapses between the notification of the change and the end of the update (i.e., $t_2 - t_1$) is not mastered. Yet low level alerts that occur during this interval are useful for the recognition of the new scenarios. Moreover, as a new rule often replaces a previous one that has been suppressed during the update, low level alerts that have been generated before this interval may also be useful. Thus it is necessary to manage an history of events on a period of time sufficient to detect attacks occurring during the changes of the architecture. In practice, a time window of several minutes is used. In the proposed solution, the size of the window is dynamically adapted: it is reduced when the workload of the correlation engine is high. At time t_2, the number of low level alerts that have been received and never analysed is a good measure to estimate the current workload of the correlation engine.

V. Assessment and Measures

In the context of the European project PANOPTESEC, partners have created an emulation environment which corresponds faithfully to the system for energy distribution managed by the main stakeholder of the project: a water and energy distribution company of Italy. The original system is based on SCADA equipments and communication protocols. Obviously, as lives and critical infrastructure are at stake, our new software components cannot be integrated and tested directly within the operational network. The emulation platform that has been realized integrates devices and software that are exactly the same as those used in the disaster recovery site of the company. In particular some used physical system are in fact cold stand-by devices located in the Disaster Recovery site. To obtain an emulated environment very similar to a recent snapshot of the real one, elements of the production environment have been combined with virtualized clones. In the emulated environment, real data generated at runtime by the devices are mixed with real SCADA traffic.

This emulated environment allows us to work on realistic data. In particular, at runtime, the other components feed our correlation engine with sets of correlation rules and a flow of low level alerts that are very representative of what would have been obtained in the real system. The flow of low level alerts is generated by the various security devices integrated in the SCADA system. As indicated before, we limit the exploitation of the attack graph to attack scenarios of length less or equal to 5. This allows us to have sets of nearly 20 thousands correlation rules. To obtain this number of rules, the generator of rules (quite close from [12]) was tuned to generate sequences without trying to merge rules with a same prefix. Moreover, as some nodes have different communication interfaces, the number of discovered topological paths (and thus the number of correlation rules) is rather high.

Note that during the PANOPTESEC project [13], another partner (Sapienza University of Rome) was in charge of developing the same services using an existing software, namely the Complex Event Processing engine Esper [14]. Their query-based tools exhibit quite similar performances when the set of rules is stable. Their query-based approach allows to consider other approximations of the scenario (for example, they can check if all the steps of a scenario occur in an order that is not the one described by a rule). Yet, using an on-shelf component appears to be less performant and flexible when the set of queries may change: a complet reset is more often unavoidable. Moreover it is sometimes more difficult to obtain information that concerns all the queries/rules. For example, in the automata-based approach, it is trivial to identify a low level alert that did not allow to advance in any scenario recognition. Memory management is also less easy to manage directly in the query-based approach.

A. Accuracy and Amount of Raised Alerts

First the accuracy of the whole intrusion detection process has been evaluated. As the emulation platform is a controlled environment, we can assert with an high level of confidence that no attack is underway as long as an attack scenario is not explicitly played by us. When the parameters (namely Max_k for the Missing Alerts and the threshold for the Ongoing Alerts) are tuned in a reasonable way, no false positive has been generated. With a few multi-step attack scenarios against the command and control centres, we study the occurrence of false negatives. When the knowledge base used both to generate the attack graph and to enrich alerts is up-to-date, all the attacks have been detected. Yet, when some IDSes are not

able to detect attack steps, false negatives may occur. Among the played attacks, one attack is a denial of service attack against high voltage nodes of the infrastructure. In a first step, an external attacker exploits SMB vulnerabilities to take the control of the file server of a first machine. Then this machine is used as a gateway. In a second step, an archive server on a second machine is compromised. At this stage, the attacker is now able to open connections with all the high voltage Front End devices. In a third step, the attacker gains control of one of them by exploiting a vulnerability of the running ftp server. From this compromised front end, the attacker performs ssh connections with the other active front ends to send shutdown commands. Communication paths from SCADA Servers to RTUs (Remote Terminal Units) are broken. The command and control system is taken offline.

Regarding the above scenario, we provide now more detailed information about the raised alerts. Our objective is to distinguish complete alerts (where all the attack steps are identified) from Ongoing Alerts (where only a prefix is identified) and from missing alerts (where one step is missing). To show the interest of the k-missing algorithm and to show that false negatives may occur if the deployment and/or the configuration of some IDSes is not satisfactory, we have considered two different settings. In one case, all the steps of the attacker have been recognized and at least one low-level alert refers to each of them. In the second case, the second step of the attack remains undetected. Results are summarized in Table III.

TABLE III: Detection of an Attack Scenario Composed of 4 Steps

Setting	Case 1 (second step detected)		
Type of Alerts	Complete & Exact	Complete with Missing	Ongoing
$Max_k = 0$	1	0	12
$Max_k = 1$	1	20	320

Setting	Case 2 (second step not detected)		
Type of Alerts	Complete Exact	Complete with Missing	Ongoing
$Max_k = 0$	0	0	0
$Max_k = 1$	0	1	45

Around 20 thousands correlation rules were specified. A complete and exact alert is raised when all the attack steps of a given scenario are identified. When a missing attack step is allowed, a complete alert is always raised. A false negative (i.e., the lack of a complete alert) only occurs in case 2 when Algorithm 1 is running (i.e., $max_k = 0$). Note that the number of Ongoing Alerts remains quite low (equal to either 0, 12, or 45) except in case 1 when $max_k = 1$ (equal to 320). Indeed in this particular case, as all the steps of the attack are observable in the flow of low level alerts, the use of the joker does not palliate the absence of a specific type of low level alert. Here the use of the joker leads to identify variants of the right attack scenario. More precisely, many Ongoing Alerts will correspond to correlation rules where the first step differs (the attacker can select different entry points) or the third step differs (the attacker can select different front end devices). In some sense, all these additional alerts are approximations of the right one: their multiplicity does not mean that they are without interests. Moreover, as each alert is explicitly tagged "Complete and Exact", "Complete with Missing" or "Ongoing", the administrator can adopt this order of priority during the analysis of the alerts already received: the many Ongoing Alerts are useful as long as no complete alert is available.

B. Performance Evaluation

The environment used for the evaluation (i.e., the machine on which the correlation engine runs) has the following characteristics: Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz, RAM 4GB.

1) Initialization Time and Updates: The time required to initialize all the data structures is linear with the number of rules (Figure 5). Only one plan is created per automaton when

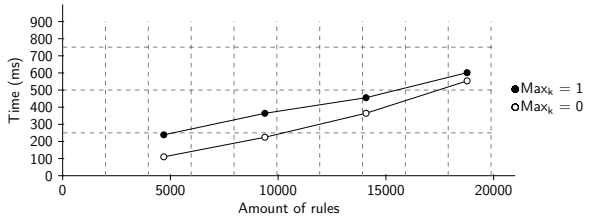

Fig. 5: Time Required to Initialized the Whole Set of Rules

$max_k = 0$. More than one plan is created when $max_k = 1$. In the worst case, at runtime, an update may lead to suppress all the previous rules and to consider a whole set of new rules. In that case, the analysis may be suspended during a period of several hundreds of milliseconds. Yet most of the time, updates are incremental and impact a much smaller number of rules. Of course, the set of rules can be spread over several machines to considerably reduce this cost.

2) Tolerated Rate of Incoming Low Level Alerts: Here we analyse the highest rate tolerated when no garbage collection is launched and no update of the set of rules is performed. Obviously, the percentage of low level alerts that corresponds to attack steps of some rules has an influence on processing times. Thus we create an artificial flow of alerts where this parameter is taken into account. Five percentages are considered (0%,10%,30%,50%,100%). Through an analysis of real logs, it seems that a realistic percentage is usually between 10% and 30%. To see the cost of generating Missing Alerts, we analyze the two cases $Max_k = 0$ (Algorithm 1) and $Max_k = 1$ (Algorithm 2). Moreover, to show the interest of distributing the rules on two machines (which analyze the same flow), we indicate the maximum rate for the complete set of rules (18767 rules) and for half (9384 rules). The results are described in Table IV.

3) Memory Consumption: Memory consumption is a real problem because the number of plans grows continuously. When $Max_k = 1$, the garbage collector approach (Figure 6) is

TABLE IV: Average Number of Analyzed Low Level Alerts per Second

	$Max_k = 0$		$Max_k = 1$	
Matching %	18767 rules	9384 rules	18767 rules	9384 rules
0%	15178	15363	14901	15017
10%	4048	4208	3981	4098
30%	1200	1300	1060	1172
50%	770	825	676	703
100%	347	708	260	485

compared with an extreme case where only the most advanced plans are kept (Figure 7).

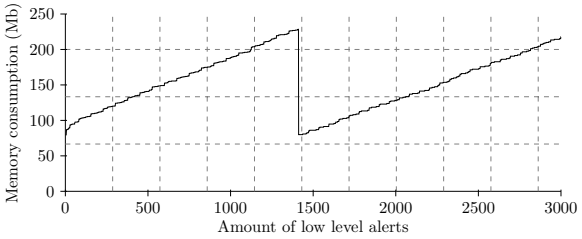

Fig. 6: Garbage with Threshold

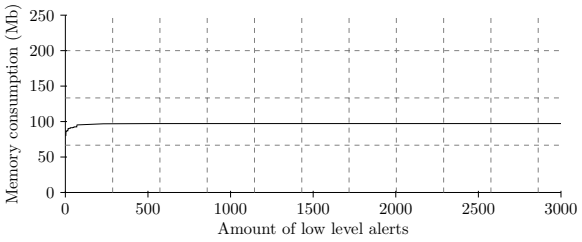

Fig. 7: Only the Most Advanced Plans are Kept

The first figure shows that new plans are continuously created while the second figure indicates that the number of most advanced plans is bounded. A mixing between the two approaches seems to be the best strategy to obtain a good tradeoff between the memory consumption and the accuracy of the analysis. If an attack occurs, it will be detected even if all the plans except the most advanced ones have been suppressed. Yet, when the alert is raised, the administrator is not informed of all the combinations of low level alerts that are matching with the scenario. For example, the administrator may have a partial view of the situation when several distinct attackers are concurrently executing the same attack. Keeping additional plans, even if they are not tagged as being one of the most advanced plans is a simple solution to tackle these extreme cases.

VI. Conclusion

In distributed systems and in particular in industrial SCADA environments, alert correlation systems are necessary to identify complex multi-step attacks within the huge amount of alerts and events. We have described an automata-based correlation engine developed in the context of a European project where the main stakeholder was an energy distribution company. In the proposed solution, a fully automated process generates thousands of correlation rules. Despite this major scalability challenge, the designed correlation engine exhibits good performances. Expected rates of incoming low level alerts approaching several hundreds of elements per second are tolerated. Moreover, the used data structures allow to quickly handle dynamic changes of the set of correlation rules.

Acknowledgment

This work has been partially supported by the European Union Seventh Framework Programme under grant agreement No. 610416 (PANOPTESEC) and by the French Ministry of Defence (DGA).

References

[1] E. Totel, B. Vivinis, and L. Mé, "A Language Driven Intrusion Detection System for Event and Alert Correlation," in *Proc. of the 19th IFIP International Information Security Conference*. Kluwer Academic, Aug. 2004, pp. 209–224.

[2] E. Godefroy, E. Totel, M. Hurfin, and F. Majorczyk, "Automatic Generation of Correlation Rules to Detect Complex Attack Scenarios," *Journal of Information Assurance and Security*, 2015.

[3] F. Valeur, "Real-Time Intrusion Detection Alert Correlation," Ph.D. dissertation, University of California, 2006.

[4] B. Morin, L. Mé, H. Debar, and M. Duccassé, "M4d4: a Logical Framework to Support Alert Correlation in Intrusion Detection," *Information Fusion*, vol. 10, no. 4, pp. 285–299, 2009.

[5] C. Michel and L. Mé, "ADeLe: an Attack Description Language for Knowledge-based Intrusion Detection," in *Proc. of the 16th International Conference on Information Security (IFIP/SEC 2001)*, 2001, pp. 353–365.

[6] F. Cuppens and R. Ortalo, "LAMBDA: A Language to Model a Database for Detection of Attacks," in *Proc. of the Third International Workshop on the Recent Advances in Intrusion Detection (RAID'2000)*, H. Debar, L. Mé, and S. F. Wu, Eds., 2000, pp. 197–216.

[7] S. T. Eckmann, G. Vigna, and R. A. Kemmerer, "STATL: An Attack Language for State-based Intrusion Detection," in *Proc. of the ACM Workshop on Intrusion Detection*, 2000.

[8] J. Goubault-Larrecq and J. Olivain, "A Smell of Orchids," in *Proc. of the 8th Workshop on Runtime Verification (RV'08)*, ser. Lecture Notes in Computer Science, M. Leucker, Ed., vol. 5289. Budapest, Hungary: Springer, 2008, pp. 1–20.

[9] R. A. Kemmerer and G. Vigna, "Hi-DRA: Intrusion Detection for Internet Security," *Proceedings of the IEEE*, vol. 93, no. 10, pp. 1848–1857, Oct. 2005.

[10] B. Schneier, "Attack Trees: Modeling security threats," *Dr. Dobb's Journal*, vol. 24, no. 12, pp. 21–29, Dec. 1999.

[11] X. Ou, W. F. Boyer, and M. A. McQueen, "A Scalable Approach to Attack Graph Generation," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2006, pp. 336–345.

[12] S. Jajodia, S. Noel, and B. O'Berry, "Topological Analysis of Network Attack Vulnerability," in *Managing Cyber Threats*, ser. Massive Computing. Springer, 2005, vol. 5.

[13] Consortium, "PANOPTESEC Project," http://www.panoptesec.eu/.

[14] EsperTech, "Esper product webpage," http://www.espertech.com/esper/.