Automatic Generation of Correlation Rules to Detect Complex Attack Scenarios

Erwan Godefroy, Eric Totel, Michel Hurfin, Frédéric Majorczyk

Automatic Generation of Correlation Rules to Detect Complex Attack Scenarios

Erwan Godefroy*†‡, Eric Totel†, Michel Hurfin‡ and Frédéric Majorczyk*
* *DGA-MI, Bruz, France frederic.majorczyk@intradef.gouv.fr*
† *Supélec, Rennes, France erwan.godefroy@supelec.fr eric.totel@supelec.fr*
‡ *Inria, Rennes, France michel.hurfin@inria.fr*

***Abstract*—In large distributed information systems, alert correlation systems are necessary to handle the huge amount of elementary security alerts and to identify complex multi-step attacks within the flow of low level events and alerts. In this paper, we show that, once a human expert has provided an action tree derived from an attack tree, a fully automated transformation process can generate exhaustive correlation rules that would be tedious and error prone to enumerate by hand. The transformation relies on a detailed description of various aspects of the real execution environment (topology of the system, deployed services, etc.). Consequently, the generated correlation rules are tightly linked to the characteristics of the monitored information system. The proposed transformation process has been implemented in a prototype that generates correlation rules expressed in an attack description language.**

***Keywords*-Security and Protection; Intrusion detection;**

I. Introduction

An alert correlation system aims at exploiting the known relationships between some elements that appear in the flow of low level notifications to generate high semantic meta-alerts. The main goal is to reduce the number of alerts returned to the security administrator and to allow a higher level analysis of the situation. However, producing correlation rules is a highly difficult operation, as it requires both the knowledge of an attacker, and the knowledge of the functionalities of all IDSes involved in the detection process. This paper focuses on the transformation process that allows to translate the description of a complex attack scenario into correlation rules. Obviously, the first phase (*i.e.*, the identification of the elementary actions that compose the attack scenario) requires the knowledge of a human expert. In this work, we assume that this specification relies on attack trees [7]. These structures are extended to identify the elementary actions that compose a multi-step attack. This hierarchical formalism allows to describe the logical relations between elementary actions and to indicate some temporal constraints between them.

From this extended attack tree, we define a transformation process for the creation of correlation rules. The sequential transformations take progressively into account various information about the execution context. Our correlation rules generator needs two kinds of information that can be provided by two kinds of independent experts. Some experts provide information about attack scenarios while the others provide information on the execution environment. Thus the areas of expertise that should be required for each of them is smaller. The initial representation of an attack has to be as compact and as general as possible while the resulting set of correlation rules has to be specific to a concrete environment. The goal is to ensure that this analysis remains valid even if the environment evolves from time to time. The expert has only to describe the main steps and characteristics of a complex attack without identifying neither the possible targeted nodes, nor the possible observations that may lead to detect that a given step of this attack has occurred.

Section II presents an overview of our approach for automatically generating correlation rules. Section III illustrates our starting point with an example. Section IV defines the notion of action tree and Section V describes the steps required to transform an action tree into a correlation tree. Section VI reviews the related works and Section VII concludes the paper.

II. From attack trees to correlation rules

The generation process is divided into five steps (See Figure 1). Among the five steps, four can be entirely automated by using data stored in a knowledge base. Only the first step requires the involvement of a human expert during the transformation process. An attack tree (provided by a human expert) is the starting point of our transformation process. This tree includes the operators defined in [1] (OR, AND, SAND). The attack scenario can be completely generic (in that case, the attack tree describes the steps of a known attack) or specific to the information system to be monitored.

The first step consists in transforming the attack tree into an action tree. An attack tree cannot be used directly as input to an automated transformation process because its description is too informal and because it refers only to goals and sub goals of the attack. The rewriting operated by the expert addresses these two problems. Broadly, the built action tree is quite similar to what has been proposed in [1]: an action is associated with each leaf node. Obviously, any implicit information must be explicit in the action tree. An action description language (detailed later in Section IV) is a cornerstone in the proposed transformation process.

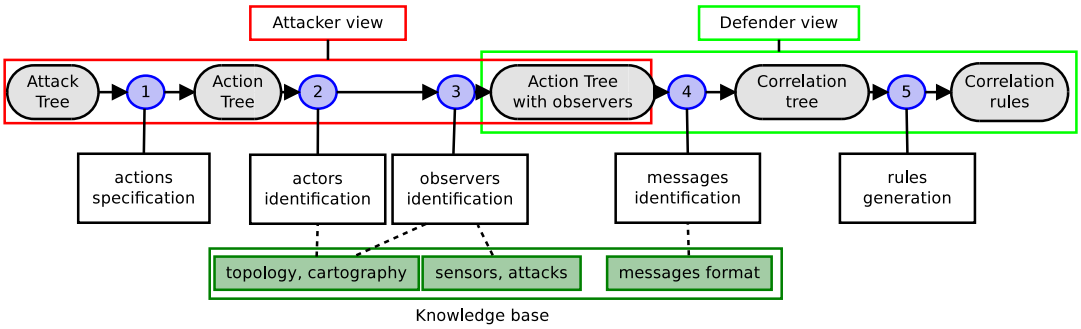


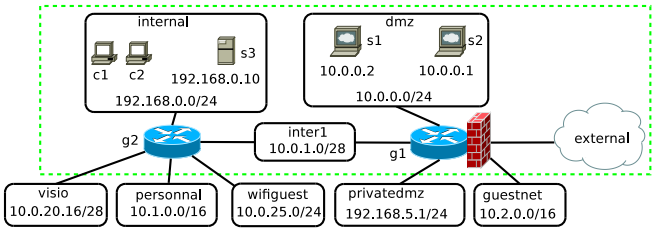
Figure 1. Transformation process of attack an tree to correlation rules: steps and used structures

The purpose of the next steps is to adapt the action tree to the specificities of the monitored information system by using information from the knowledge base. The second step aims at identifying the actors of the different actions (i.e., the hosts that can be the source or the target of an action). For each action of the action tree and for each unspecified host, the cartography (installed softwares and services) provided by the knowledge base is used to find the possible hosts. For example, if the action is a basic attack's step, only the hosts where a vulnerable process is running are considered. The third step aims at identifying observers that have the opportunity to observe the specified actions. Again the discovery process relies on the knowledge base which contains information about the different sensors, NIDS and HIDS. More details are provided in the Subsection V-B. The fourth step consists in transforming the action tree with observers into a correlation tree that describes the observable events that can be used to detect the whole attack scenario. IDSes and sensors will respectively raise alerts or send messages to the correlation system. This step is fully described in the Subsection V-C. During the fifth step of the process, the correlation rules are generated so that they can be included in the available correlation system. This step involves mostly syntax translations.

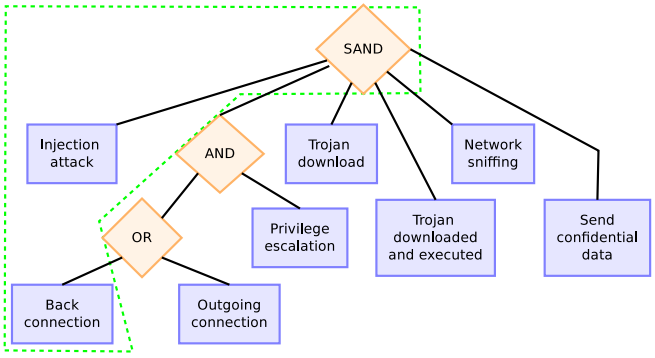
The generation process explained above relies on a knowledge base describing the monitored system and the deployment of the sensors and IDSes in this system. This description takes into account the topology of the network (the sub-networks where the nodes are placed, their addresses), the cartography (which services, processes and softwares are running on each node) and the different sensors monitoring the system. Some ontologies-based models ([4], [2]) have been proposed as a knowledge base and can fulfill partially these requirements. We choose to use a knowledge base that extends M4D4 ([4]) because most of the objects and relationships we have to model are already taken into account. This framework is built upon a Prolog engine, which enables the definition of facts and rules in the same language.

III. Introducing a practical example

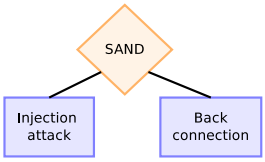
A. System architecture



(a) Network topology



(b) Attack scenario to steal confidential information



(c) Subset of a global attack

Figure 2. Topology and attack trees

In order to illustrate our approach, we propose to apply each step in the context of an example. The reference system is shown on Figure 2a. For the sake of simplicity, our example is limited to the sub-part surrounded with the dotted rectangle (with the minimal number of machines required to run the scenario). The gateway firewall *g1* makes the

public DMZ (identified by *dmz* in Figure 2a) accessible from the outside but does not authorize any connection from the outside world or from the DMZ to the *internal* subnetwork. However, the internal network and the DMZ can access the internet. The DMZ contains two hosts s1 and s2 that run different services. Both hosts run web servers that are accessible from the internet. They also act as file servers by hosting FTP services. These services are only accessible from the internal network. The internal network is composed of two clients c1 and c2 and a file server s3 which is only accessible from the internal network. The role of s3 consists in storing confidential files.

We can define the topology and cartography of this system with facts defined in an extension of M4D4. In our implementation, such information is represented in the form of Prolog facts and can be accessed through queries. For example, the IP address of the node *s1* is defined with *node_address(s1, ipv4(10,0,0,1))*.

B. Attack Tree Example

We suppose that an external attacker wants to steal some valuable information stored in the internal server s3. One possible scenario could consist in the following steps: as the attacker can only access the subnetwork *dmz*, its attack strategy consists in compromising one of the DMZ server in order to place a trojan horse then one user in the internal network will access this malicious file, and thus will compromise his machine in the internal network. The attack tree describing one possible multi-step attack is illustrated in Figure 2b. In order to illustrate our approach, we will focus on a small part of this attack tree, which consists in a possible attack to upload the initial trojan to the public DMZ server. This attack consists in a sequence of two actions: a code injection that is followed to a connection to the attacker machine. This small attack is described by the attack tree on Figure 2c.

IV. Action Tree

This section focuses on how an action tree can be derived from an attack tree by describing (1) each elementary action and (2) how these actions can be linked together.

A. Action Description Language

The attacker actions are specified using an action description language that describes the action and its actors (source and target). Each action is defined by its name. The possible names are identified in the following paragraph. An action can require to define the source and target of the action. This information constitutes the attributes of the action and are defined in the following.

Action Naming Convention: Elementary actions of an attack tree can be split in two categories: they can be attacks, which means that they match a known attack class, or they can be normal actions which are considered malicious only within the context of the scenario as a whole. As a consequence, we choose an appropriate naming convention for each of these two categories.

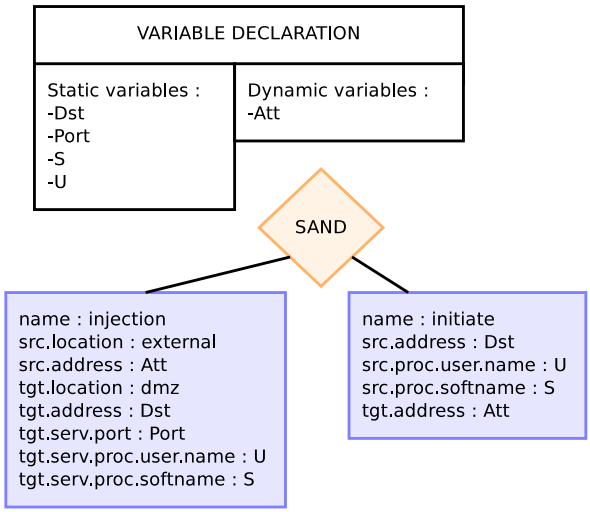
As actions that refer to attacks should refer to known categories of attacks, we need an attack taxonomy in order to name attacks. Among the taxonomies which have been proposed, CAPEC[1] (Common Attack Pattern Enumeration and Classification) seems to be a suitable choice. Indeed, this taxonomy takes into account about 400 attack patterns organized in about 70 different categories. This taxonomy is built from 15 main attack mechanisms (from Data Leakage Attacks to Supply chain Attacks). Each of these mechanisms is then subdivided into other attack mechanisms until it reaches elementary attack patterns that compose the leaves of the CAPEC tree. During the definition of the knowledge base, this organization allows to express general mechanisms of attacks which may encompass several more detailed attack patterns. During the specification of an action, the chosen name of an attack action has to correspond to the name of the common ancestor attack mechanism that includes potentially all the attack patterns the attacker can perform. This enables us to specify an action with a variable level of precision. However, choosing a too general attack mechanism may lead to false positives, whereas choosing a too specific attack pattern may lead to false negatives. In our example, the first leaf is an attack action labelled *Injection attack*. Injection is one of the root attack mechanism provided by CAPEC. It is sub-classed in 17 different attack categories (from remote code inclusion to SQL injection). We can decide to stop here and choose *injection* for the name of the action. But if the context gives more restrictions about the possible attacks, we can for example choose the injection sub-class *command injection* and consequently reject all other types of injections.

Actions which are not attacks by themselves are supposed to be normal system actions. Common system actions are referenced in the CEE[2] (Common Event Expression) taxonomy. CEE goal is to standardize the representation of logs. Consequently, it provides system primitives including a list of elementary actions which can be performed in a system. Contrary to CAPEC, the action taxonomy is reduced to a simple list of terms. In our example, the second leaf describes an outgoing connection, which can be described by the action *initiate*.

Attributes: The description of an action should allow to decide if the action is observable on the system, and how this observation can be performed. In order to answer these questions, it is necessary to know at different observation levels (network, system, application) if the action is visible.

In practice, to answer this question, it is necessary to know if an attack is remote (and involves packets on the

[1] http://capec.mitre.org/
[2] http://cee.mitre.org



(a) Action tree

Figure 3. Action description language

network), or if it is local (and is locally observable). To provide this information, we decided to define two attributes: source and target. The source and the target have locations and IP addresses on the network. If these attributes are equal, the attack is local to a node. Moreover, an action has always an impact on an entity on the system: this entity can be a file, a service listening on a port, or a process. A file is completely determined by its file name. A process has a name, and runs with the rights of a user (that is defined by his name and his user id). This set of attributes is sufficient to know what level of the system is targeted by the attack. It is important to note that each attribute of the action description has a type. If one of these attributes is represented by a variable, this variable inherits from the type of the attributes it represents.

B. Variable or constant attributes

The attack scenario consists of several actions with different kinds of attributes: some of them are constant and loosely coupled with the system characteristics and are known before the attack occurs. Some others are in the defender zone and are strongly coupled with the system topology and configuration. Finally, some attributes are under the control of the attacker, which means that their values cannot be determined before an attack occurs and can change for each attack instance. These several kinds of attributes conduct to the definition of the nature of each attribute: it is either constant or variable and variable attributes can be divided in two categories: static and dynamic.

Constant Attributes: An attribute is constant when its value is constant for all instances of the described attack (e.g., the network in which the web servers are located). This constant has the type of the attribute and can be used to define a value which is known to be an attack invariant. In the example Figure 3a, the *target.location* of the left action is always the dmz, as it is the only part of the network that is reachable by the external attacker.

Variable Attributes: An attribute must be represented by a variable when its value can vary depending on the instance of the attack. We can then distinguish two categories of variables: the variable whose possible values can be enumerated knowing the system configuration (topology, cartography, etc.), and the variables that take values completely independently of the system. The first category of variables is called *static* variable, and the second category is called *dynamic* variable. The category of each variable is defined in the declaration part of the variables.

In the example Figure 3a, the *target.address* of the action is one of the two servers in the DMZ. As a consequence, the *target.address* attribute is a variable whose values is in the set {10.0.0.1, 10.0.0.2}. This set can be completely enumerated using the knowledge of our network. This attribute is thus a *static* variable. On the contrary, the value of the *source.address* attribute of the left action cannot be guessed uniquely by relying on the knowledge of the system topology. This attribute will take a value only during the execution of the attack: it will be the IP address of the attacker machine. This attribute is thus a *dynamic* variable.

Relations between actions: A variable that has been declared in the variable declaration part can be used in several actions if necessary, as long as it is used for attributes of the same type. In the example Figure 3a, the variable *Att* (whose value is the IP address of the attacker machine) describes both the *source.address* of the left action and the *target.address* of the right action. This is a way to define that the value of the attribute *source.address* of the left action is equal to the attribute *target.address* of the right action.

V. Automatic generation of correlation tree

A. Taking into account the target system

The purpose of this second step is to translate the generic action tree defined in IV-A into an action tree specific to the targeted system.

Action tree transformations: During this step all attributes parameters with static variables will be identified thanks to suitable requests to the knowledge base. These requests rely on a connection between the action attributes and the predicates that describe the different actors. This instantiation is subdivided in two operations. The first operation produces the list of all the possible trees given the information available in the knowledge base and the second operation merges these trees into a single one.

The first operation produces a list of trees whose action parameters previously defined with a static variable will be instantiated. Each time a static variable can receive more than one value, a new tree is created. As a consequence, each output tree reflects one possible specific attack tree in our current system. Then, given that all these trees refer to

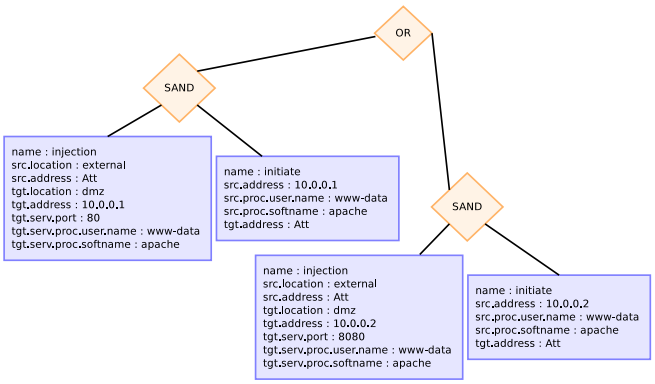


Figure 4. Actors identification

the same attack scenario, we consider that only one attack tree should include them. Therefore, a new common *OR* root node is added to link all these trees.

Illustration: Given the action tree of the previous section, the system will request the address, service port and process related to a running service for all nodes located in the DMZ. Moreover, when the two actions are remote actions, the filtering rules of the gateway will be taken into account. Indeed, the gateway allows external nodes to only access ports *80* and *8080* respectively of servers *s1* and *s2*. Then, the services (and the related softwares and users information) associated to each of these ports for each server will be identified. Given the cartography of our example, two servers (*s1* and *s2*) will be identified as potential actors. Consequently, the static variables will take these specific values, resulting in two trees (one dedicated for each server) that will be merged as shown in Figure 4.

B. Who can observe the actions?

This step focuses on selecting suitable observers for each actor. An observer is any device able to provide information about system or network events. They can be a simple sensor or a sophisticated IDS. During this step, a new parameter will be added to each action. This parameter will associate each action to its set of potential observers. A potential observer of a specific action is an observer for which the topological visibility and the functional visibility are compatible with the action specification. The topological visibility of an observer refers to the set of actors it can monitor and its functional visibility refers to the type of action it can detect based on its configuration. These definitions extend those provided in [4].

The topological visibility of an observer can be a network or a node. Consequently, an action involving only one node can be potentially only observed by a HIDS or a local sensor, while actions involving two different nodes can be observed by local and remote sensors. The selection of all observers providing a suitable topological visibility is achieved in two stages: first, all suitable host-based sensors are selected. Then, the suitable network sensors are selected based on the possible routes between the source and target of the action.

Then, we suppose that an observer can detect specific actions. The functional visibility is hence related to the *name* action field describing the type of action. The functional visibility is modeled by a double association. Each sensor is linked to a detection configuration and each detection configuration is linked to the type of attack or actions it can detect. Hence, the selection of observers given their functional visibility can be split in two cases. When the action refers to an attack class, all sensors able to detect one of the sub-classes or parent-classes of this action in the CAPEC taxonomy are selected. Indeed, a sensor can produce an alert referring to an attack which is a subclass of the specified action or it can produce a more generic alert. When the action is a common event (whose name is issued from CEE), all sensors whose configuration can observe the action are selected. These selections are illustrated in Figure 5.

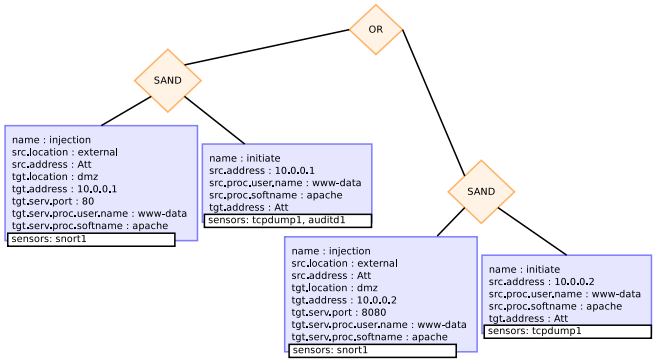


Figure 5. Action tree enriched with observers

C. From action tree to correlation tree

The goal of this section is to explain how the structure defined in Figure 5 can be transformed into a correlation tree, i.e., a structure describing only observables (called messages in our context) that can be used to detect an attack against the system. Three mechanisms are required to describe this step: how we describe messages produced by sensors, how an action is transformed into messages for a given sensor and how messages are managed when several sensors monitor the same action.

Message model: In our model, each message is composed of two parts: the message format and a set of fields. The format identifies the real format in which the message will be generated at run-time. The set of fields expresses the information that the message actually contains.

Converting actions to messages: This step takes into account the subset of attributes that messages generated by a given sensor can really include. As an example, a network sensor can generate messages containing IP source address, IP destination address, source and destination port values,

but these messages will probably not include process or user related information. In addition, a given sensor can potentially produce different messages for a given action. For example, an IDS can provide different signatures for detecting sub-classes of a specific attack class. In this case, the possible messages are joined by an *OR* operator, meaning that we assume the action occurs if at least one of the messages is raised. When several sensors can detect the same action, we consider the possible messages for each of them. Then, they are joined together with an *OR* operator. As a consequence, the action will be successful if at least one sensor detects it.

Illustration: Figure 6 shows the correlation tree resulting from the identification of the sensor generated messages. The tree is non-symmetrical because the servers *s1* and *s2* are monitored by a different set of sensors. In our case, two Snort signatures match the injection action and have respectively the signature id 22063 and 22064.

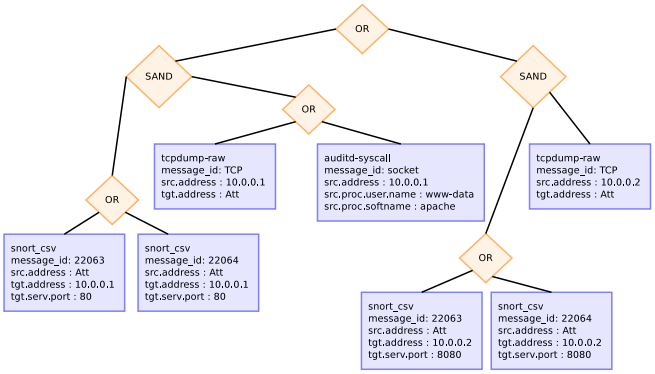


Figure 6. Correlation tree

VI. Related work

In the explicit alert correlation field, works focus mainly on the efficiency of detection algorithms and less in the configuration of these correlation systems with attack scenario. However some techniques we used in our approach to generate correlation rules are similar to those of systems built for different purpose.

The works [8] and [5] use a model of the supervised system (including network services, host reachabilities and known vulnerabilities). However, their purpose is not to generate rules but to check the accuracy of received alerts at run time. They also propose different solutions to model alerts. [8] defines alert type but without relying on an existing taxonomy. The approach [5] proposes a way to map alert generated by Snort with their defined attack scenario. It consists in a manual mapping between Snort signatures and Nessus vulnerability identifiers.

Some works on correlation use an attack graph as input attack scenario ([3], [6]). These graphs are automatically generated but take only known vulnerabilities referenced in the system into account. In general, method relying on attack graph can only take into account attack paths exploiting known and automatically identified vulnerabilities, whereas our approach allows a more global description without needing to know each vulnerability on each host. From our point of view, our approach is complementary, as attack tree could be built from information extracted from attack graphs.

VII. Conclusion and Future work

In this paper, we introduce a method for generating correlation rules from an attack scenario and a knowledge base. In order to reach this goal, we have defined an action description language to specify each elementary action and we have built an extension of the knowledge base M4D4. The benefits of this approach are twofold: first, it decouples the required expert knowledges for creating correlation rules in three low coupled domains (attack scenario specification, system cartography and sensor modeling). Then the evolutions of the system are more easily taken into account. Indeed, suitable changes involving cartography or sensors have to be made in the knowledge base to reflect the system evolutions. Then, these changes will be taken into account during a re-generation of the correlation rules.

References

[1] S. A. Çamtepe and B. Yener. Modeling and detection of complex attacks. In *Proc. of the 3rd Int. Conf. on Security and Privacy in Communications Networks*, France, 2007.

[2] G. G. Granadillo, Y. B. Mustapha, N. Hachem, and H. Debar. An ontology-based model for siem environments. In Springer, editor, *ICGS3 '11 : 7th Int. Conf. in Global Security, Safety and Sustainability*, volume 99, pages 148–155, 2012.

[3] S. Jajodia and S. Noel. Topological vulnerability analysis: A powerful new approach for network attack prevention, detection, and response. *Indian Statistical Institute Monograph Series*, 2007.

[4] B. Morin, L. Mé, H. Debar, and M. Duccassé. M4d4: a logical framework to support alert correlation in intrusion detection. *Information Fusion*, 10(4):285–299, 2009.

[5] S. Noel, E. Robertson, and S. Jajodia. Correlating intrusion events and building attack scenarios through attack graph distances. In *ACSAC*, pages 350–359, 2004.

[6] S. Roschke, F. Cheng, and C. Meinel. A new alert correlation algorithm based on attack graph. In *Computational Intelligence in Security for Information Systems*. 2011.

[7] B. Schneier. Attack trees: Modeling security threats. *Dr. Dobb's Journal*, 24(12):21–29, 1999.

[8] D. Xu and P. Ning. Alert correlation through triggering events and common resources. In *ACSAC*, 2004.

This work was partially funded by the European project Panoptesec (FP7-GA 610416).