M2D2: A Formal Data Model for IDS Alert Correlation

Benjamin Morin[1], Ludovic Mé[2], Hervé Debar[1], and Mireille Ducassé[3]

[1] France Télécom R&D, Caen, France
{benjamin.morin|herve.debar}@rd.francetelecom.com,
[2] Supélec, Rennes, France
Ludovic.Me@supelec.fr,
[3] IRISA/INSA, Rennes, France
ducasse@irisa.fr

Abstract. At present, alert correlation techniques do not make full use of the information that is available. We propose a data model for IDS alert correlation called M2D2. It supplies four information types: information related to the characteristics of the monitored information system, information about the vulnerabilities, information about the security tools used for the monitoring, and information about the events observed. M2D2 is formally defined. As far as we know, no other formal model includes the vulnerability and alert parts of M2D2. Three examples of correlations are given. They are rigorously specified using the formal definition of M2D2. As opposed to already published correlation methods, these examples use more than the events generated by security tools; they make use of many concepts formalized in M2D2.

1 Introduction

Intrusion detection today faces four major challenges. Firstly, the overall number of alerts generated is overwhelming for operators. Secondly, too many of these alerts falsely indicate security issues and require investigation from these operators. Thirdly, the diagnosis information associated with the alerts is so poor that it requires the operator to go back to the original data source to understand the diagnosis and assess the real severity of the alert. Finally, the number of undetected attacks (a.k.a false negatives) must be reduced in order to provide an appropriate coverage for the monitoring of the environment.

Alarm correlation is the key to the first three challenges: it consists in reducing and conceptually interpreting multiple alarms such that new meanings are assigned to these alarms [1]. In order to reduce false negatives, heterogeneous analyzers are often used to get multiple points of view of the attacks. Multiplying analyzers also multiplies alerts. Alert correlation is therefore all the more essential.

At present, alert correlation techniques do not make full use of the information that is available. For example, they tend to only use the events generated

by security tools. We argue that alert correlation must take at least four information types into account: information related to the characteristics of the monitored information system, information about the vulnerabilities, information about the security tools used for the monitoring, and information about the events observed. Indeed, an organization information system (IS) is made of a network of hosts which potentially exhibit vulnerabilities. These vulnerabilities are exploited by attackers. Security tools are required to monitor the IS in order to locate events of interest, identify attacks and generate alerts.

In this paper, we propose a model, called M2D2, which supplies the above four types of information. It provides concepts and relations relevant to the security of an information system. M2D2 is formally defined, following Mc Hugh's suggestion that further intrusion detection significant progress will depend on the development of an underlying theoretical basis for the field [2].

The model has nevertheless a practical ground. Indeed, a relational database implements parts of it, and the corresponding tables are automatically generated from data sources. IDS and vulnerability scanners fill the database with events, vulnerabilities and products tables come from the ICAT [3] database.

Three examples of correlations are given. The first one aggregates alerts referring to the same host, even when the host name is not provided by the alert attributes. The second example identifies the hosts vulnerable to an attack occurrence. The third example aims at reducing false positives by analyzing the reaction or the absence of reaction of the security tools which are able to detect the cause of the alert. These examples are rigorously specified using the formal bases of M2D2. Moreover, they illustrate the need to work on the four types of information.

Two sets of related work address the modeling of information for intrusion detection : the NetStat Network Fact Base of Vigna and Kemmerer [4, 5] and the Intrusion Reference Model (IRM) of Goldman et al. [6]. The Network Fact Base is formally defined but it is not comprehensive, only the topology of information systems and some of the events are modeled. IRM is quite comprehensive but we are not aware of any formalization.

As it is illustrated in the examples of correlations, the model must be both comprehensive and formally defined. Indeed, if it is not comprehensive enough, all the useful properties cannot be expressed; if it is not formally defined, the properties are not as rigorous as they must be. In the following, we reuse the concepts of Vigna's formal model [7], and extend the formalization to encompass the four types of information needed. The main contribution of this article is therefore a formal model of data relevant for alert correlation. This model is richer than the existing formal models.

Section 2 presents the concepts of the model and the relations and functions which link concepts together. Section 3 presents some correlation methods based on the model. Section 4 discusses related work. The last section concludes on the contributions of this article.

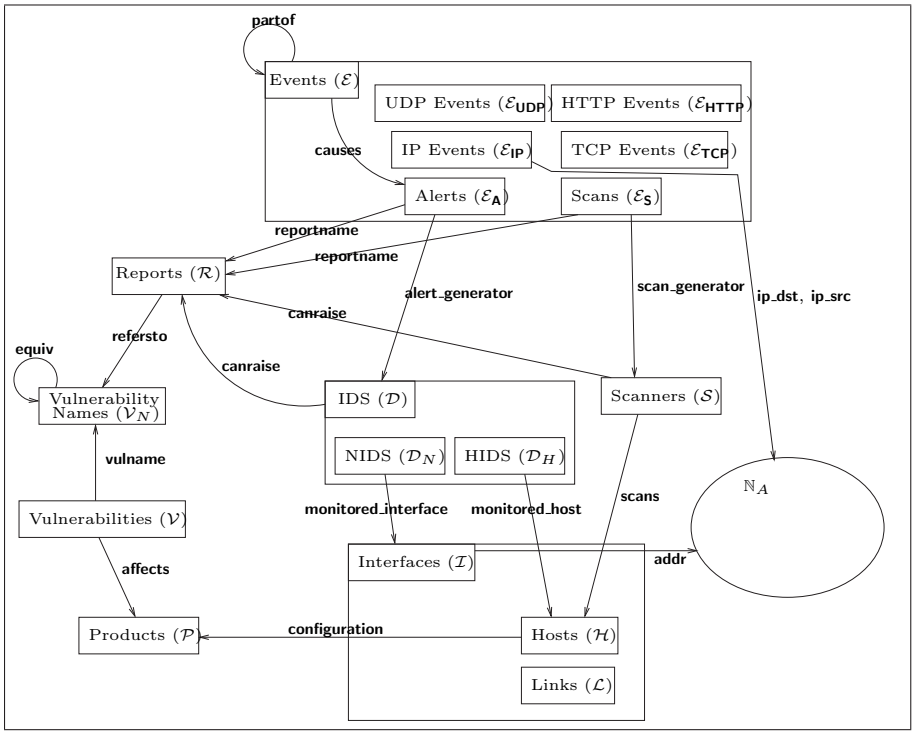

Fig. 1. M2D2 scheme

2 M2D2: Concepts and Relations

M2D2 integrates the four main types of information relevant to security identified in the introduction: IS characteristics, vulnerabilities, security tools, and events. The global schema of M2D2 can be found in figure 1. It summarizes the main concepts and relations described thereafter.

In the remainder of this section the four types of information are described in turn. For each type of information, the formal definition can be found in framed figures, and an informal description of the concepts can be found in the text. Every time a relation is informally discussed, its identifier is indicated in bold (**equiv** for example) so that its formal definition can be easily found in the corresponding frame.

2.1 Notations

For formalization, notations of the Z and B formal methods are used [8]. The precise definitions of all the operators used in this article can be found in figure 2. Let us briefly recall here the basic ones. The set of all the relations between two sets s and t is noted $s \leftrightarrow t$. A relation represents a zero-or-possibly-many to

Name	Syntax	Definition	Conditions
Set of relations	$s \leftrightarrow t$	$\mathcal{P}(s \times t)$	
Inverse relation	r^{-1}	$\{y, x \mid (y, x) \in t \times s \wedge (x, y) \in r\}$	$r \in s \leftrightarrow t$
Domain	$\mathbf{dom}(r)$	$\{ x \mid x \in s \wedge \exists y \cdot (y \in t \wedge (x, y) \in r)\}$	$r \in s \leftrightarrow t$
Range	$\mathbf{ran}(r)$	$\{ y \mid y \in t \wedge \exists x \cdot (x \in s \wedge (x, y) \in r)\}$	$r \in s \leftrightarrow t$
Composition	$r \ ; q$	$q \circ r$	$r \in s \leftrightarrow t \wedge q \in t \leftrightarrow u$
Identity	$\mathbf{id}(s)$	$\{ x, y \mid (x, y) \in s \times s \wedge x = y\}$	
Restriction	$u \lhd r$	$\{ x, y \mid (x, y) \in r \wedge x \in u \}$	$r \in s \leftrightarrow t \wedge u \subseteq s$
Image	$r[w]$	$\mathbf{ran}(w \lhd r)$	$r \in s \leftrightarrow t \wedge w \subseteq s$
Partial functions	$s \nrightarrow t$	$\{ r \mid r \in s \leftrightarrow t \wedge (r^{-1} \ ; r) \subseteq \mathbf{id}(t) \}$	
Total functions	$s \rightarrow t$	$\{ f \mid f \in s \nrightarrow t \wedge \mathbf{dom}(f) = s \}$	
Injections	$s \rightarrowtail t$	$s \rightarrow t \cap s \nrightarrow t$	
Surjections	$s \twoheadrightarrow t$	$s \nrightarrow t \cap s \rightarrow t$	
Bijections	$s \rightarrowtail\!\!\!\!\!\twoheadrightarrow t$	$s \rightarrowtail t \cap s \twoheadrightarrow t$	

Fig. 2. Notations

zero-or-possibly-many correspondence. The set of all partial functions between s and t is noted $s \nrightarrow t$. A partial function represents a zero-or-possibly-many to zero-or-one correspondence. The set of all total functions between s and t is noted $s \rightarrow t$. A total function represents a one-or-possibly-many to zero-or-one correspondence. The set of all partial (respectively total) injections between s and t is noted $s \rightarrowtail t$ (resp. $s \rightarrowtail t$). A partial (respectively total) injection represents a zero-or-one to zero-or-one (resp. one to zero-or-one) correspondence. The set of all partial (respectively total) surjections between s and t is noted $s \twoheadrightarrow t$ (resp. $s \twoheadrightarrow t$). A partial (respectively total) surjection represents a zero-or-possibly-many to one (resp. one-or-possibly-many to one) correspondence. The set of all bijections between s and t is noted $s \rightarrowtail\!\!\!\!\!\twoheadrightarrow t$. A bijection represents a one to one correspondence.

The distinction between relations and functions, as well as the injective property, have a major impact on implementation. An injective function can be implemented straightforwardly by an array. A non-injective function can be implemented by an array but the contents of the array may be a pointer to a list of values. In general, a relation which is not an injective function is best implemented by a database table. The partial versus total distinction has a major impact on the interpretation of the data. Indeed, if a function is partial some of the values may be undefined. A relation which is not a function may also be partial. It can be specified by the fact that the domain is only included in, and not necessarily equal to, the origin set ($\mathbf{dom}(r) \subseteq s$). Bijections are the easiest to implement and handle, unfortunately, very few relations are bijections in reality. In the following, we will therefore rigorously define the characteristics of the defined relations.

Definitions
$\mathcal{I}$ is the set of network interfaces.
A network is a hypergraph on $\mathcal{I}$, i.e is a family $N = (E_1, E_2, \ldots, E_m)$, where $E_i \subset \mathcal{I}$ is called an edge. Edges are divided in two subsets, $\mathcal{H}$ and $\mathcal{L}$
$\mathcal{H} \subseteq \mathcal{P}(\mathcal{I})$ is the set of hosts, and
$\mathcal{L} \subseteq \mathcal{P}(\mathcal{I})$ is the set of links.
$\mathcal{N}$ is the set of host names in the information system.
$\mathbb{N}_A = \{0, \ldots, 2^{32} - 1\}$ is the set of IPv4 address.

Functions & Relations
$\textsf{addr} \in \mathcal{I} \rightarrowtail \mathbb{N}_A$
$\textsf{hostname} \in \mathcal{H} \rightarrow \mathcal{N}$

Properties
Hosts and links both partition interfaces, i.e

$$\bigcup_{i=1}^{p} H_i = \mathcal{I} \text{ and } H_i \cap H_j = \emptyset \ (\forall i, j \in \{1, 2, \ldots, p\} \text{ and } i \neq j)$$
$$\bigcup_{i=1}^{q} L_i = \mathcal{I} \text{ and } L_i \cap L_j = \emptyset \ (\forall i, j \in \{1, 2, \ldots, q\} \text{ and } i \neq j)$$

Comments
A *hub* is a type of link, for example.
See figure 4 for a network example. Interfaces are dots, hosts are circles and links are lines between interfaces.
Hosts should uniquely be identified by their host name, however two distinct hosts can have the same host name, thus **hostname** is not a bijection.

Fig. 3. M2D2 IS topology model

2.2 Information System's characteristics

Modeling information systems (IS) is a difficult task because IS are increasingly complex and dynamic. Alert correlation must take into account information system specificities. As an example, alert correlation requires evaluating the success probability of an attack, which can be achieved by comparing the target configuration with the vulnerability's known prerequisites.

As mentioned earlier, an IS characteristics depends on both the topology (network of hosts) and the products used.

Topology The topology model is formally defined in figure 3. We adopt the TCP/IP networks topology formal model proposed by Vigna [7]. Vigna suggests to model a network with an hypergraph. Nodes of the graph are the network interfaces, edges are subsets of the interfaces set[1]. Edges are divided into two kinds of components, hosts and links. From that physical network model, an IP network is modeled with an injective function which maps interfaces with IP addresses (**addr**).

[1] in standard graphs, edges are nodes couples; in an hypergraph, edges are subsets of the nodes

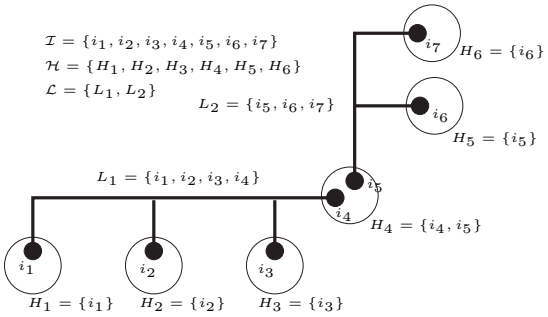

Fig. 4. A sample network

In addition to Vigna's model, M2D2 also includes other topological information, such as the mapping between hosts and host names (**hostname**). This information is useful for alert correlation purposes because IDS either give the former or the latter depending on their data source. We also rely on the IANA list for mapping services names to port numbers.

M2D2 information system model does not contain all relevant concepts yet. Missing concepts (users and files for example) must be integrated inside M2D2 in the future in order to reach comprehensiveness.

From a practical point of view, some automated tools exist to build and maintain the network topology and product database [2]. Network topology can also be manually built with graphical tools.

Products Products are formally defined in figure 5. In M2D2, a product is simply a logical entity that is executed by a host. A product is a quadruplet (`vendor_id`, `product_id`, `version_id`, `type`). `vendor_id` and `product_id` respectively represent the vendor name (**vendor**) and the product name (**prodname**). The version number (**version**) is problematical because there is currently no canonical representation of software product versions, and version notations are heterogenous (generally an alphanumeric string). That is why **version** is a partial function. For alert correlation purposes, it is useful to compare the version number of two products, for example to evaluate the vulnerability of a host. There is currently no solution to this problem. The product set may be any set provided with a total order relation. We suggest to use $\mathbb{N}^p$. The total order relation ($<$) is given in figure 5. Each product belongs to a class (**prodtype**) which may be one of $\{$`OperatingSystem`, `xxxServer`, `LocalApp`, `other`$\}$, where `xxxServer` stands for any server-like application, i.e an application which listens to a port. `xxx` is to be replaced by a protocol name like `http`, `ftp`, `snmp`.

[2] SNMP protocol (`http://www.ietf.org/rfc/rfc1157.txt`), NetSaint tool for Unix (`http://www.netsaint.org`), System Management Server for Windows (`http://www.microsoft.com/smserver/`)

Definitions

$\mathcal{P}$ is the set of products,
VN is the set of product vendor names,
PN is the set of product names,
$PV = \mathbb{N}^p$ is the set of product version ($\mathbb{N}$ is the set of integers),
$PT = \{\texttt{OperatingSystem}, \texttt{LocalApp}, \texttt{httpServ}, \texttt{ftpServ}, \ldots, \texttt{other}\}$
is the set of product types.

Functions & Relations

$$\begin{aligned}
\textbf{vendor} &\in \mathcal{P} \nrightarrow VN \\
\textbf{prodname} &\in \mathcal{P} \rightarrow PN \\
\textbf{version} &\in \mathcal{P} \rightarrow PV \\
\textbf{prodtype} &\in \mathcal{P} \nrightarrow PT \\
\textbf{configuration} &\in \mathcal{H} \rightarrow \mathbb{P}(\mathcal{P}) \\
<_{\in} &\; PV \leftrightarrow PV
\end{aligned}$$

Properties

$\forall v_1, v_2 \in \mathbb{N}^p, v_1 < v_2 \iff \exists j | \forall i, 1 \leq i \leq j, v_1[i] = v_2[i] \wedge v_1[j] < v_2[j]$

Comments

Not all products have a vendor attribute, thus **vendor** is a partial function.
Two distinct versions of the same product are modelized by two $\mathcal{P}$elements, i.e **version**is not a relation.

Fig. 5. M2D2 product model

Services, operating systems, local applications are products which run on hosts. The set of products running on a host is called a *configuration*. Formally speaking, a host configuration is a subset of products.

2.3 Vulnerabilities

Vulnerabilities are defined in figure 6. Most of current attacks consist in exploiting vulnerabilities that exist in entities of an IS. Shirey defines a vulnerability as a flaw or weakness in a system design, implementation, or management that could be exploited to violate the system security policy [9]. Shirey's vulnerable system corresponds to an M2D2 product. However, vulnerabilities sometimes do not refer to a single product: a host actual vulnerability may depend on the combined presence of several products (a web server on a given operating system for instance); a protocol design error may involve vulnerabilities in several implementations of the protocol (see SNMP's vulnerabilities `CAN-2002-0012` for instance (a.k.a BugTraq id 4088)). Arlat et al. define a vulnerability as a latent error present on a target [10], a target corresponding to a M2D2 host. However, it is not practical to model a vulnerable entity that way because each vulnerability would require a dedicated relation or function between the host set and the vulnerabilities set. In M2D2 a vulnerability affects a configuration (**affects**). In that way, a host is vulnerable if its configuration is a superset of one vulnerable configuration.

Definitions
$\mathcal{V}$ is the set of CVE and CAN vulnerabilities
$\mathcal{V}_N$ is the set of vulnerability names

Functions & Relations

$$\begin{aligned}
\textbf{affects} &\in \mathcal{V} \leftrightarrow \mathcal{P}(\mathcal{P}) \\
\textbf{req} &\in \mathcal{V} \rightarrow \{\texttt{Remote, RemoteUser, Local}\} \\
\textbf{con} &\in \mathcal{V} \rightarrow \{\texttt{CodeExec, DoS, Info}\} \\
\textbf{vulname} &\in \mathcal{V} \rightarrowtail \mathcal{V}_N \\
\textbf{equiv} &\in \textbf{vulname}[\mathcal{V}] \leftrightarrow \mathcal{V}_N
\end{aligned}$$

Properties
$\textbf{vulname}[\mathcal{V}]$ is the set of CVE and CAN vulnerability names.

Comments
affects is a relation because a vulnerability may affect several configurations and a configuration may be affected by many vulnerabilities.

Fig. 6. M2D2 vulnerability model

The vulnerability set is built from the ICAT vulnerability database. Moreover, ICAT uses the CVE/CAN[3] naming convention proposed by Mann [11] to uniquely identify the vulnerabilities. The Mitre proposes a list of name equivalences between CVE and other vulnerability names. This list is used inside M2D2 to cluster vulnerability names which refer to the same vulnerability (**equiv**). Ideally, **equiv** should be an equivalence relation, but it is not in reality. As Mann notices, the mapping between vulnerability names and CVE names is seldom one-to-one, so a non-CVE name may be equivalent to more than one CVE names. As a result, from one non-CVE vulnerability name, it is possible to get many CVE vulnerabilities.

Modeling vulnerabilities actually requires that a classification is established. As a matter of fact, no dictionary of terms for describing alerts exists [6], but alerts often refer to a known vulnerability. As a result, the information found in vulnerability databases remains the only source of information for describing alerts. One of the ICAT database strengths is that many attributes are used to classify vulnerabilities. For alert correlation purpose, we have chosen the two criteria which are described thereafter. Criteria can easily be extracted from the ICAT database, they are common to all classifications and are relevant for alert correlation, while remaining simple. For example, correlation may require to detect if a privilege elevation occurred, or if several kinds attacks have been attempted.

- Access requirements (**req**) for the vulnerability to be exploited
  - `Local`: the attacker needs a physical local access to the target,
  - `RemoteUser`: the attacker needs a remote access to the target with user-level privileges,

[3] `http://cve.mitre.org/`

Definitions

$\mathcal{D}$ is the set of IDS,
$\mathcal{S}$ is the set of vulnerability scanners,
$\mathcal{R}$ is the set of tool-specific report names (alert or scan),
$\mathcal{D}_N$ is the set of NIDS
$\mathcal{D}_H$ is the set of HIDS

Functions & Relations

$$\begin{aligned}
\textsf{meth} &\in \mathcal{D} \rightarrow \{\texttt{Misuse, Anomaly}\} \\
\textsf{data} &\in \mathcal{D} \rightarrow \{\texttt{HostBased, NetworkBased}\} \\
\textsf{monitored_interface} &\in \mathcal{D}_N \rightarrow \mathcal{I} \\
\textsf{monitored_host} &\in \mathcal{D}_H \rightarrow \mathcal{H} \\
\textsf{scans} &\in \mathcal{S} \leftrightarrow \mathcal{H} \\
\textsf{canraise} &\in \mathcal{D} \cup \mathcal{S} \leftrightarrow \mathcal{R} \\
\textsf{refersto} &\in \mathcal{R} \leftrightarrow \mathcal{V}_N
\end{aligned}$$

Properties

$\mathcal{D}_N = \textsf{data}^{-1}[\{\texttt{NetworkBased}\}]$
$\mathcal{D}_H = \textsf{data}^{-1}[\{\texttt{HostBased}\}]$

Fig. 7. M2D2 security tools model

  - `Remote`: the attacker only needs a remote access to the target.
- Consequence (**con**) of a successful exploit of the vulnerability:
  - `DoS`: the exploit results in a denial of service.
  - `InformationGathering`: the exploit results in information disclosure. For example, directory traversals and file reading fall under this category.
  - `CodeExecution`: the exploit results in the execution of arbitrary code. For example, buffer overflows and race conditions fall under this category.

2.4 Security Tools

Security tools model is formalized in figure 7. Vulnerabilities can be detected either when they are exploited by an attacker or simply when they are latent. IDS detect the former, vulnerability scanners detect the latter. IDS and vulnerability scanners are security tools.

Although they are part of the topology, security tools need to be explicitly described in a security model like M2D2. Security tools are event producers (see 2.5). Any security tool type should be integrated into M2D2. In the current model, IDS are qualified by their analysis method (**meth**), either misuse-detection or anomaly detection. Other security tools, like firewalls, will be included in future versions of M2D2.

IDS detect, identify and generate alerts about active attacks in real time. We distinguish two kinds of IDS according to their data source: HIDS (Host-based IDS) and NIDS (Network-based IDS) (**data**). This dichotomy is based on the *topological visibility* of an IDS: the *topological visibility* of an IDS refers to the set of network components an IDS can monitor. NIDS (Network-based IDS)

read streams of events on a network interface (**monitored_interface**), thus they can monitor many hosts depending on their location in the network. Whereas a HIDS (should it be system-based or application based) monitor a single host (**monitored_host**), vulnerability scanners do vulnerability assessments on several hosts, thus their topological visibility is modeled with the relation between a scanner and a host (**scans**).

We also introduce the *operational visibility* of an IDS which refers to the ability of an IDS to detect vulnerability exploits according to its configuration[4]. This is modeled by a relation (**canraise**) between an IDS or a scanner and a report name. A report name is tool-specific, i.e the names of the reports are the ones that are used by the tool that generates the alerts. Report names are linked to one or more vulnerability identifiers (**refersto**). It is therefore transitively possible to know which vulnerability exploits can be detected by an IDS.

The correspondence between report names and vulnerabilities are found, for instance, in IDS signature files for misuse IDS.

It has to be noted that all report names do not refer to a vulnerability. As a matter of fact, all attacks are not vulnerability exploits (port scanning is such example). Some IDS can detect an attack but not recognize it as the exploit of a given vulnerability, either because the attack is not referenced yet or because the IDS cannot do so (case of the anomaly IDS).

2.5 Events, Alerts, and Scans

The event formal model is described in figure 8.

Both vulnerability scanners and IDS generate messages about the existence or about an ongoing exploit of a vulnerability. The former is called a *scan*, and the latter an *alert*. In the case of alerts, security tools should also generate the *events* which led them to generate alerts.

Arlat et al. define an event as the instantaneous image of a system [10]. In other words, an event is the manifestation of some activity the format of which is discussed in this section. In the intrusion detection field, event and alert are generally two distinct concepts [12]. An event is a low level entity (TCP packet, system call, syslog entry, for example) from which an analysis is performed by a security tool. An alert is a message from an analyzer signaling that one or more events of interest have been detected. We say that an alert is *a kind of* event, since it reflects the state of an IDS. Such a definition is compliant with the previous definitions and enables to elegantly model the fact that alerts produced by an IDS may be the events from other tools' point of view. In other words, current IDS are low level event consumers and alert producers but future IDS may be alert consumers and high level alerts producers.

M2D2 models alerts, scans, and the following event types: IP events, TCP events, UDP events, HTTP events and HTTP log events. We restricted the event types to the set of event types understood by most current IDS. However, M2D2

[4] e.g the set of activated signatures for a misuse IDS

Definitions

$\mathcal{E}$	is the set of event instances,
$\mathcal{E}_{\mathsf{A}}$	is the set of alerts,
$\mathcal{E}_{\mathsf{S}}$	is the set of scans,
$\mathcal{E}_{\mathsf{IP}}$	is the set of IPv4 events,
$\mathcal{E}_{\mathsf{TCP}}$	is the set of TCP events,
$\mathcal{E}_{\mathsf{UDP}}$	is the set of UDP events,
$\mathcal{E}_{\mathsf{HTTP}}$	is the set of HTTP events,
$\mathcal{E}_{\mathsf{LOG}}$	is the set of web servers log events.

Properties

$\mathcal{E}_{\mathsf{A}}, \mathcal{E}_{\mathsf{IP}}, \mathcal{E}_{\mathsf{TCP}}, \mathcal{E}_{\mathsf{UDP}}, \mathcal{E}_{\mathsf{HTTP}}, \mathcal{E}_{\mathsf{LOG}}$ partition $\mathcal{E}$

Functions & Relations

$$
\begin{aligned}
\textsf{tstamp} &\in \mathcal{E} \to \mathbb{N} \\
\textsf{partof} &\in \mathcal{E} \leftrightarrow \mathcal{E} \\
\textsf{alert\_generator} &\in \mathcal{E}_{\mathsf{A}} \to \mathcal{D} \\
\textsf{reportname} &\in \mathcal{E}_{\mathsf{A}} \cup \mathcal{E}_{\mathsf{S}} \to \mathcal{R} \\
\textsf{scan\_generator} &\in \mathcal{E}_{\mathsf{S}} \to \mathcal{S} \\
\textsf{scan\_host\_target} &\in \mathcal{E}_{\mathsf{S}} \to \mathcal{H} \\
\textsf{scan\_port\_target} &\in \mathcal{E}_{\mathsf{S}} \to \mathbb{N} \\
\textsf{causes} &\in \mathcal{E} \leftrightarrow \mathcal{E}_{\mathsf{A}}
\end{aligned}
$$

Comments

Alerts can cause other alerts, therefore $\mathbf{dom}(\textsf{causes}) \subset \mathcal{E}$, not $\mathcal{E} - \mathcal{E}_{\mathsf{A}}$
$\mathbf{ran}(\textsf{causes}) \subsetneq \mathcal{E}_{\mathsf{A}}$ because alerts may have no cause.

Fig. 8. M2D2 model of events

has to be extended with other event types such as operating system level events and other network application protocol events, with their respective attributes.

The common attribute of all events is the timestamp (**tstamp**). The IDMEF [12] uses three distinct timestamps: `detecttime` (attack occurrence date), `createtime` (attack detection date), `analysertime` (alert generation date). Those three distinct timestamps depend on the nature of the M2D2 events: a M2D2 IPEvent (cf p. 12) timestamp is a `detecttime`. An alert (cf p. 11) timestamp is a `createtime` or a `analysertime`.

To model the aggregative nature of events, we introduce the **partof** relation. This kind of relation models the fact that an alert can aggregate (encapsulate) several other alerts or that an HTTP request is part of an HTTP log entry. This relation may be refined in the future if necessary.

In the remainder of this section, we give some details on some of the event types (alerts, scans, and HTTP log events). Others are self-explanatory (the reader may refer to the relevant RFCs).

Alerts Alerts are formalized in figure 8.

Each alert has a single report name (**reportname**). Alerts are generated by IDS (**alert_generator**).

As with IDMEF definition, alerts are caused by the occurrence of events which are the manifestation of an attack (**causes**). Note that alerts may not be linked to a *causal* event. This does not mean that an alert has no cause, but that the causal information is not available (the IDS generating the alert does not give this information, for instance). Alerts may also have more than one causes. Finally, an event can be the cause of several alerts because:

- it can be shared by several alerts generated by distinct IDS, or
- IDS may generate several alerts for one single event. For example, the HTTP request
  `GET /cgi-bin/phf?/etc/passwd` may cause three alerts: one concerning the presence of `/etc/passwd` in the URL, one concerning the presence of `/cgi-bin/phf` and one concerning the success/failure of the request.

In other existing alert formats, *source* and *target* concepts are attributes of the alerts [12, 13]. Source and target implicitly refer to the source and target of an attack. As events reflect the steps of an attack, we claim that the source and the target concepts are hold by the events themselves. For IP events, the source and the target are respectively the source IP address and the destination IP address. Port numbers play that role for TCP events. The target information may also be deduced from the location where an event was detected. This is the case for all the events raised by HIDS, since IDS only monitor the host they are running on.

Scans Scan is a shorthand for "vulnerability scan". A scan is a vulnerability assessment provided by a vulnerability scanner (Nessus[5] for instance) against a specific host.

Like alerts, scans have a tool-specific report name (**reportname**). Scans are generated by scanners (**scan_generator**).

There are two major differences between scans and alerts. Firstly, scans are not caused by events, since scans do not detect active vulnerabilities exploits. However, it should be noted that to perform security tests, scanners generate intrusive traffic which can cause IDS alerts. Alert correlation should take this information into account.

The second difference is that while alert target information is carried by low level events, the target is explicitly carried by the scan. Vulnerability assessments are made against a host (**scan_host_target**) on a given port (**scan_port_target**).

IPEvents, TCPEvents, UDPEvents, HTTPEvents These event types are modeled in figure 9.

In Vigna's model, a message is a triple (i_s, i_d, p), where $i_s, i_d \in \mathcal{I}$ are respectively the source interface and the destination interface which belong to the same link. p is the payload of the message.

[5] `http://www.nessus.org`

Definitions

$\mathcal{U}$ is the set of URL which have been raised by security tools.

Functions & Relations

$$
\begin{array}{rrll}
\textbf{ip\_src}, \textbf{ip\_dst}, \textbf{idt} & \in & \mathcal{E}_{\textsf{IP}} & \rightarrow \mathbb{N}_A \\
\textbf{ipayload} & \in & \mathcal{E}_{\textsf{IP}} & \nrightarrow \mathcal{E}_{\textsf{TCP}} \cup \mathcal{E}_{\textsf{UDP}} \\
\textbf{sp}, \textbf{dp} & \in & \mathcal{E}_{\textsf{TCP}} \cup \mathcal{E}_{\textsf{UDP}} & \rightarrow \mathbb{N} \\
\textbf{seq}, \textbf{ack} & \in & \mathcal{E}_{\textsf{TCP}} & \rightarrow \mathbb{N} \\
\textbf{tpayload} & \in & \mathcal{E}_{\textsf{TCP}} \cup \mathcal{E}_{\textsf{UDP}} & \rightarrowtail \mathcal{E}_{\textsf{HTTP}} \\
\textbf{httpmeth} & \in & \mathcal{E}_{\textsf{HTTP}} & \rightarrow \{\texttt{GET},\texttt{POST},\texttt{HEAD}\} \\
\textbf{status} & \in & \mathcal{E}_{\textsf{HTTP}} & \rightarrow \mathbb{N} \\
\textbf{url} & \in & \mathcal{E}_{\textsf{HTTP}} & \rightarrow \mathcal{U}
\end{array}
$$

Comments

ip_src, **ip_dst**, **idt** Resp. source address, destination address and identification number of an IPv4 event,

sp, **dp** Resp. source port and destination port numbers of a TCP or UDP event,

seq, **ack** Resp. sequence number and acknowledgement number of a TCP event,

httpmeth Method of an HTTP request,

status Status of an HTTP request,

url URL of an HTTP request.

ipayload is a partial function because IP Event payload may not be available and thus is not provided by the security tool.

Fig. 9. M2D2 IP Events, TCP Events, UDP Events and HTTP Events

An IP datagram is a sequence of messages whose payload is a triple (a_s, a_d, p'), where $a_s, a_d \in \mathbb{N}$ are respectively the source and destination addresses of the IP datagram. The messages in the sequence represent the route of the datagram.

An IP session is a sequence of IP datagrams and represents a conversation between two hosts.

An UDP datagram is an IP datagram whose payload is a couple $(p_s,\ p_d)$, where p_s,$p_d \in \mathbb{N}$ are respectively the source port and the destination port of the UDP datagram. An UDP session is a couple of UDP datagrams (request and response).

Basically, a TCP segment is an IP datagram whose payload is a tuple $(p_s$,p_d,s,a,F,$p'')$, where p_s,p_d,s,$a \in \mathbb{N}$ are respectively the source port, the destination port, the sequence number and the acknowledgement number, F is a set of flags and p'' is the payload of the TCP segment. A TCP session is an IP session with constraints over the datagram attributes which are described in Vigna's model.

In the same way, a M2D2 TCP event (resp. an UDP event) is a TCP segment (resp. an UDP datagram) in Vigna's model.

We do not model sessions in M2D2 yet because most current NIDS do not work at the session level. However, extending M2D2 with sessions is straightfor-

ward since the M2D2 events are conceptually identical to the concepts found in Vigna's model.

An IP event is linked to its content with the **ipayload** function. In the same way, TCP or UDP event are linked to their content (an application protocol like HTTP for instance) with the **tpayload** function.

HTTP Events An HTTP event models an HTTP interaction between a client and a server. HTTP event attributes potentially reflect a complete HTTP interaction. Thus, we use the **httpmeth**, **url**, and **status** functions. Note that if only a part of the interaction may be analyzed (that is the case for NIDS, which may only have access to a part of the interaction), then a special value is assigned to the event attribute.

HTTP Log events Log events correspond to entries in HTTP server logs. An HTTP server log entry contains an HTTP event, and an identifier of the requester (either its IP address or its host name). That mix of information coming from two different protocol layers raises a modeling problem. We thus had to introduce a specific kind of events, the HTTP Log events. That kind of events reflect the contents of a HTTP server log.

3 Alert correlation using M2D2

A first step towards alert correlation is alert aggregation. It consists in grouping alerts following various criteria. In this section, we propose three relevant examples of aggregation methods which use many M2D2 concepts. As these concepts are defined in an unambiguous way, the aggregation methods are rigorously expressed. M2D2 contribution to alert correlation resides in the use of relations between vulnerabilities and topology, between topology and security tools, as well as between security tools and vulnerabilities to model alert aggregation methods.

3.1 Aggregation of alerts referring a single host

The goal of the **common_target** aggregation function (cf figure 10) is to group alerts referring to a given target host. This set is made of two subsets: one contains the alerts generated by HIDS running on the host, and the other contains alerts generated by NIDS having detected malicious IP events destined to the host.

As HIDS monitor a single host (cf figure 7, **monitored_host**), the set of alerts referring to a host is equal to the set of alerts generated by the HIDS monitoring the host. We call this aggregation function **hids_target** (cf figure 10).

An intermediate relation is required to uniquely identify a host from a network interface (this is a function because hosts partition the interface set, cf figure 10). We call this function **belongs_to_host** (cf 16).

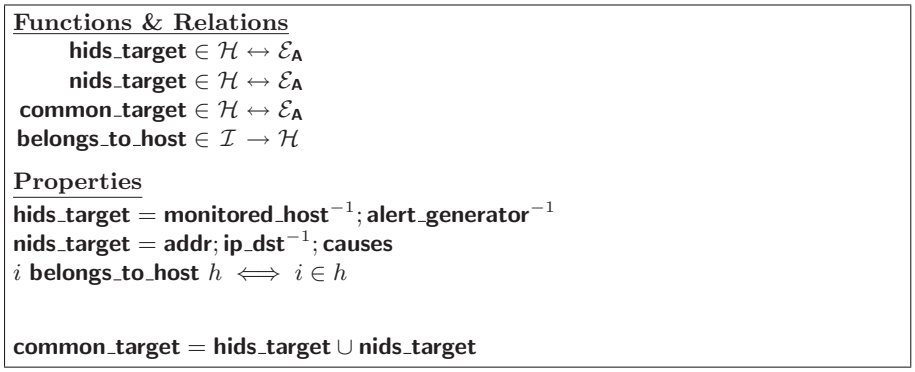

Fig. 10. The **common_target** aggregation function referring the same target host

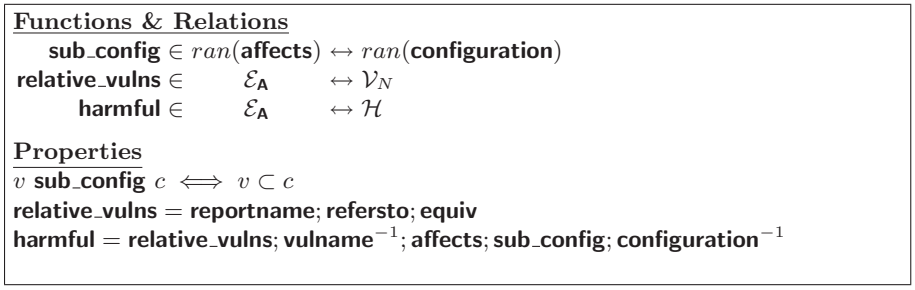

Fig. 11. The successful attacks identification function

NIDS monitor an interface which has access to a stream of network events, some of which being destined to the host under consideration. The aggregation function, **nids_target** (cf figure 10) consists in aggregating all the alerts caused by IP events whose IP destination address maps to one of the host interfaces.

3.2 Identification of the hosts vulnerable to an attack occurrence

The successful attack identification function, called **harmful** (cf figure 11), applies to a single alert and gives the set of hosts on which the attack may work. In other words, it gives information about the success of the attack.

An attack may be successful if one of the configurations affected by the vulnerability which is exploited by the attack really *exists* on some hosts in the information system. For example, an attack exploiting a ProFTP vulnerability against a WU-ftp server may not work. Thus the attack severity is low. That does not mean that the attack should not be reported (it is still an attack attempt), but its severity may be lowered.

A vulnerability $v \in \mathcal{V}$ exists on a host if the host configuration is a superset of at least one of v vulnerable configurations. Thus, we introduce the **sub_config**

(cf figure 11) relation, which maps all the vulnerability configurations with all the host configurations. **sub_config** provides all the vulnerabilities exhibited by a given host, and $\textbf{sub\_config}^{-1}$ provides all the hosts which exhibit a given vulnerability. Building and updating the **sub_config** relation in batch mode is required for efficiency because evaluating existing vulnerabilities in real time would be prohibited.

We introduce the **relative_vulns** relation which applies to an alert and maps the vulnerability names the alert report name refers to. From an alert relative vulnerability set, we get the corresponding CVE vulnerability subset, and then the vulnerable configurations set is extracted. Thanks to the **sub_config** relation, we get the set of vulnerable host configurations. Lastly, applying $\textbf{configuration}^{-1}$ to the host configuration set provides the set of hosts which are vulnerable.

3.3 Detecting false positives

In this section we present a correlation function which is a way to detect false positives. When an IDS generates an alert, it is relevant to check if all the IDS which were able to detect the events causing the alarm did generate an alert too. When an IDS generates an alert, a simple sanity check is to verify that others able to both process the events and create the alert did so. If this constraint is verified, then the alert likelihood is reinforced. If not, the proper behavior of all IDSes should be checked to decide whether the one sending the alert is providing the wrong diagnosis or whether the others are misbehaving.

Let us consider an alert $a \in \mathcal{E}_{\mathbf{A}}$. We first need to identify the alerts which are *similar* to a. That is, we need to know which alerts are caused by the same set of events. Once we have the set of similar alerts, we can know which set of IDS raised them. Finally, we have to compare this set with the set of IDS which were able to detect the events (the ability of an IDS to detect events and generate alerts depends on its topological visibility and its operational visibility (cf section 2.4)).

In the following, we first describe two methods to identify similar alerts. We then show how to get the set of IDS which did react. Lastly, we propose a function to get the *potentially reactive IDS* set (i.e the set of IDS which could react to an attack).

Alerts similarity In Howard's classification [14], an *attack* is a series of *actions* taken by an attacker to achieve an unauthorized result. Events result from these actions. As it has been described in section 2, one or more alerts may result from every each event. Determining the likelyhood of an alert can be achieved by enumerating the IDS which also generated an alert for a single *action* (be it from an attack or from a legitimate action which is believed malicious). We call alerts caused by a single action *similar alerts*. Several ways to aggregate alerts are possible, for example:

- aggregating alerts caused by the same event
- aggregating alerts referring to the same vulnerability

Definitions

$\stackrel{ip}{=} \in \mathcal{E}_{\mathsf{IP}} \leftrightarrow \mathcal{E}_{\mathsf{IP}}$

$$\forall i_1, i_2 \in \mathcal{E}_{\mathsf{IP}},\ i_1 \stackrel{ip}{=} i_2 \iff \begin{cases} \mathbf{ip\_src}(i_1) = \mathbf{ip\_src}(i_2) \wedge \mathbf{ip\_dst}(i_1) = \mathbf{ip\_dst}(i_2) \\ \qquad\qquad \wedge\ \mathbf{idt}(i_1) = \mathbf{idt}(i_2) \\ \qquad\qquad \wedge\ |\mathbf{tstamp}(i_1) - \mathbf{tstamp}(i_2)| < \varepsilon \\ \qquad\qquad \text{OR} \\ \exists i_3 \in \mathcal{E}_{\mathsf{IP}}, \text{ such that } i_1 = i_3 \wedge i_2 = i_3 \end{cases}$$

$\mathbf{sim\_alert\_ip} \in \mathcal{E}_{\mathsf{A}} \leftrightarrow \mathcal{E}_{\mathsf{A}}$

Properties

$\stackrel{ip}{=}$ is an equivalence relation.

$\mathbf{sim\_alert\_ip} = (\mathbf{causes}^{-1}; \stackrel{ip}{=}; \mathbf{causes})$

$\mathbf{sim\_alert\_ip}[\{a\}]$ is the set of alerts which are caused by the same IP event as a.

Fig. 12. Alerts caused by the same IP event

- aggregating alerts caused by events belonging to the same TCP/IP session
- aggregating alerts on a temporal relation basis

In the remainder of this section, we illustrate the first two examples of similar alerts aggregation methods.

Aggregating alerts through events Several alerts can be generated from a single event either because more than one triggers within the same IDS matches the event (a trigger is a condition fulfillment leading an IDS to generate an alert), or because more than one IDS have access to the event itself. Thus, alerts caused by the same events should be aggregated. We call events causing an alert causal events. It should be noted that two alerts do not need to have the same set of causal events to be aggregated: alerts can be aggregated if they have at least one causal event in common. For example, a suspicious HTTP request can be detected by a NIDS (A) and by an application-based IDS (B). A alert is caused by an HTTP event and the underlying network layers event. B alert is caused by an HTTP event, which is part of a web server log occurrence. Thus, A and B do not have the same set of causal events; the HTTP event is the only one they have in common. However, A alert and B alert are aggregated because they have one causal event in common.

As every alert comes along with its own causal event set, we need to define a relation to compare events. Such a relation strongly depends on the timestamp of the events. Although distinct IDS alert timestamps (IDMEF `analysertime`) may differ, the timestamp of the causal events (IDMEF `detecttime`) should be very close. Two events cannot have the same timestamps as clock synchronization is impossible to achieve. This is why we impose two events to have close

$\mathbf{sim\_alert\_vuln} \in \mathcal{E}_\mathbf{A} \leftrightarrow \mathcal{E}_\mathbf{A} \cup \mathcal{E}_\mathbf{S}$
$\mathbf{sim\_alert\_vuln} = \mathbf{relative\_vulns}; \mathbf{refersto}^{-1}; \mathbf{reportname}^{-1}$

Fig. 13. Alerts referring to the same vulnerability

$\mathbf{reactive\_ids} \in \mathcal{E}_\mathbf{A} \leftrightarrow \mathcal{D}$
$\mathbf{reactive\_ids} = (\mathbf{sim\_alert\_ip} \cup \mathbf{sim\_alert\_vuln}); \mathbf{alert\_generator}$

Fig. 14. Reactive IDS

timestamps (in figure 12, ε is a constant whose value is close to the maximum gap existing between two synchronized clocks).

We recursively define the comparison relation between IP events, noted $\overset{ip}{=}$ (cf figure 12). Two IP events are equal if they have the same attributes (destination address, source address, identification number) and close timestamps or if there is another IP event which is equal to the two others. A recursive definition is required here otherwise $\overset{ip}{=}$ would not be an equivalence relation due to the time constraint. Equality relations for other types of events can be built in the same way.

Applying the **causes** (figure 8) relation to an $\overset{ip}{=}$ equivalence class gives the set of alerts which are caused by the same IP event. The **sim_alert_ip** (cf figure 12) is a function which applies to an alert, and gives a set of alerts which are caused by the same IP event (if the alert is caused by an IP event, otherwise this is the empty set).

Aggregating alerts through vulnerability As a second example, we propose a way to aggregate alerts generated by IDS which not only use distinct data sources, but also use distinct alert naming conventions. Thus, we propose to group alerts which refer to the same vulnerability (figure 13).

Let us consider an alert $a \in \mathcal{E}_\mathbf{A}$. The set of a similar alerts is composed of the alerts whose relative vulnerabilities (defined in figure 11) are the same as a.

Reactive IDS Reactive IDS are the IDS which have generated an alert because of a single action. Actions are on the attacker's side so we do not have access to this concept. We only have access to the manifestations of actions, i.e events and alerts. Thus, for a given alert $a \in \mathcal{E}_\mathbf{A}$, the reactive IDS are the IDS which have generated alerts similar to a. In the preceding paragraphs, we introduced two examples of similar alert aggregation functions. The reactive IDS can be obtained by simply applying **alert_generator** (cf figure 8) to the similar alerts. We call this relation **reactive_ids** (cf figure 14).

$$\mathbf{oper\_able} \in \mathcal{E}_\mathbf{A} \leftrightarrow \mathcal{D} \cup \mathcal{S}$$
$$\mathbf{oper\_able} = \mathbf{relative\_vulns}; \mathbf{refersto}^{-1}; \mathbf{canraise}^{-1}$$

Fig. 15. Operationnaly-able IDS

$$\mathbf{topo\_able\_hids} \in \mathcal{E}_\mathbf{A} \leftrightarrow \mathcal{D}_H$$
$$\mathbf{topo\_able\_hids} = \mathbf{alert\_generator}; \mathbf{monitored\_host}; \mathbf{monitored\_host}^{-1}$$
$$\mathbf{topo\_able\_nids} \in \mathcal{E}_\mathbf{A} \leftrightarrow \mathcal{D}_H$$
$$\mathbf{topo\_able\_nids} = \mathbf{causes}^{-1}; \mathbf{ip\_dst}; \mathbf{addr}^{-1}; \mathbf{belongs\_to\_host}; \mathbf{monitored\_host}^{-1}$$
$$\mathbf{topo\_able} \in \mathcal{E}_\mathbf{A} \leftrightarrow \mathcal{D}_H$$
$$\mathbf{topo\_able} = \mathbf{topo\_able\_hids} \cup \mathbf{topo\_able\_nids}$$

Fig. 16. Topologically-able IDS

Potentially reactive IDS A potentially reactive IDS is an IDS which is both topologically and operationally able (see section 2.4) to detect an attack. Topological and operational ability both apply to alerts: given an alert $a \in \mathcal{E}_\mathbf{A}$, operationally-able IDS (resp. topologically-able IDS) are IDS which are operationally-able (resp. topologically-able) to generate an alert similar to a.

Given an alert $a \in \mathcal{E}_\mathbf{A}$, the operationally-able IDS set is composed of the IDS which can raise a report which refers to one of a relative vulnerabilities. This function is formalized in figure 15.

Evaluating the topologically-able IDS faces two major pitfalls. The first one is that the topological visibility of an HIDS is limited to a single host, whereas the topological visibility of an NIDS is not even limited to a static set of hosts. An NIDS having access to a network stream of events depends on the route followed by the datagrams. Thus, it is generally not possible to say that an NIDS has access to the datagrams of such attack without knowing *a priori* the route taken by the datagrams, and this information is not available.

The second pitfall is that an attack may concern more than a single host. In other words, a set of events may be considered malicious only because of the multiplicity of the targets (network probes are such examples). In such a case, although an IDS is operationally able and topologically able to detect some *part of* an attack, it may not generate an alert.

Moreover, if an alert $a \in \mathcal{E}_\mathbf{A}$ is generated by an HIDS, no reliable conclusion can be sketched about NIDS topological ability to detect the corresponding attack: if the attack is launched locally, it has no network-side effect ; if it is launched remotely, no information about the route taken by the datagrams is available

Thus, the set of topologically-able IDS is currently limited to the set of HIDS. We propose two functions to obtain topologically HIDS set for a given alert. The first one is called **topo_able_hids** and applies to alerts generated by HIDS and gives HIDS which monitor the same host. The second relation, called

$\mathsf{able} \in \mathcal{E}_{\mathsf{A}} \leftrightarrow \mathcal{D}_H$
$\mathsf{able} = \mathsf{topo\_able} \cap \mathsf{oper\_able}$

Fig. 17. Potentialy reactive IDS

topo_able_nids, applies to alerts generated by an NIDS and gives HIDS which monitor the destination host of the underlying IP event which causes the alert (cf figure 16).

Lastly, potentially reactive IDS are both topologically-able and operationally-able IDS. As the we are currently limited to topologically-able HIDS, potentially reactive IDS is also limited to potentially reactive HIDS. The corresponding relation is in figure 17.

3.4 Discussion

M2D2 provides an expressive tool to build complex correlation methods. For example, the reduction of false positives described in section 3.3 could have been hardly designed without such a rich and formal model. Moreover, the three aggregations presented above have all illustrated how powerful and easy it is to compose relations. Aggregations themselves could be composed further. For example, the first two aggregations can be composed to create another relevant aggregation relation ($\mathsf{harmful}^{-1}; \mathsf{common\_target}$). Indeed, applying $\mathsf{harmful}^{-1}$ to a given host gives alerts whose related attacks may be successful. When used in conjunction with the **common_target** function, we know which attacks may work on a given host.

The formal specification also helps detect a number of problems. For example, the relation sub_config figure 11 has been introduced to ensure type consistency between the composed relations. This has a practical impact on the implementation: either an intermediate table has to be created or, at correlation time, the sub-configuration will have to be checked.

4 Related Work

Formal models As already mentioned in the introduction, the closest related work are the NetStat Network Fact Base (NNFB) of Vigna and Kemmerer [4, 5] and the Intrusion Reference Model (IRM) of Goldman et al. [6]. Both the NetStat Network Fact Base and the Network Entity Relationship Database (NERD) of IRM are based on Vigna's network formal model [7], and so are the topolopy and network events of M2D2. The concept of product in M2D2 is similar to the concept of services in NNFB and NERD. The security tools formalized in M2D2 are mentioned in NNFB and IRM but they are not precisely specified. As far as we know, no model includes the vulnerability and alert parts of M2D2. On the other hand, IRM includes a security goals database. It captures security policies set by the administrators. Evidence about violations of the security policy are

then aggregated. This feature, not yet formalized, could be added in M2D2 in the future. Finally, as mentioned in section 2.5, there are no system level events modeled in M2D2. We are currently working on the addition of such events.

Another piece of work related to ours is the one of the IETF Intrusion Detection Working Group (IDWG) [12]. The Intrusion Detection Message Exchange Format (IDMEF) has been proposed to provide a common format of alerts [12]. The alert specification in M2D2 is compatible with IDMEF, IDMEF-compliant alerts can be received and processed within the M2D2 framework without loss of semantical information. The scope of M2D2 is larger than IDMEF, to take into account all sources of information relevant to information security, and as such simplifies the IDMEF model through the removal of classes with similar semantic content while conveying the same information through aggregation relationships. This is in particular the case of the SOURCE and TARGET classes, replaced by the notion of node while the *source* and *target* information are carried by aggregation relationships.

Implicit models for correlation Another aspect of the related work is correlation. In order to perform fusion or correlation, researchers in the field have worked out implicit models dedicated to their correlation algorithms. The objective of M2D2 is to have a solid modeling foundation that allows multiple correlation tools and algorithms to be implemented and tested.

In [15, 16], Cuppens and Miège base their work on fusion on the IDMEF data model [12], implemented as Prolog predicates. As such, their model is simpler than M2D2, which means that less information is available for correlation purposes. The sole reliance on intrusion-detection alerts makes it impossible to assess the relevance of these alerts with respect to the monitored information system.

Cuppens and Miège suggest in [16] to correlate alerts by using attack scenarios specified in the Lambda language [17]. Such a language allows correlations between events belonging to a same attack scenario to be expressed, and could be used on top of M2D2.

The Tivoli RiskManager tool [13] is a correlation platform for security events provided by information systems. Originally developed for the correlation of intrusion-detection alerts, it now also correlates information from firewalls and antivirus systems. The data model used by Tivoli RiskManager was the source for the first version of IDMEF, but IDMEF has evolved since and the constraints of the Tivoli environment are such that it would be extremely difficult to implement the exact IDMEF model. Also, events from the three information sources (intrusion-detection systems, firewalls and antivirus systems) are not related together as is the case in M2D2. As such, the correlation algorithm used in RiskManager has to develop hypotheses concerning the relationships between events, creating relationships based on superficial characteristics of the events.

5 Conclusion

In this paper, we introduced M2D2, a formal information model designed for security information representation and correlation. The model includes the four types of information we think crucial to allow rich alert correlations: information system characteristics, vulnerabilities, security tools, and events and alerts. The model is a platform to provide answers to three important issues in intrusion-detection today, the sheer number of alerts provided, the quality of these alerts and the precision of the final diagnosis proposed to operators.

As far as possible, M2D2 reuses models proposed by others, for instance the Vigna's topological model. The first contribution of M2D2 is thus the integration of multiple interesting and relevant concepts into a unified framework. When needed, we extended these reused model to encompass the four types of useful information.

Moreover, M2D2 is formally defined, following Mc Hugh's suggestion that further intrusion detection significant progress will depend on the development of an underlying theoretical basis for the field. This formalism is the second contribution of M2D2, ensuring that processing of security information and in particular alert correlation is anchored on a rigorous model representing the information being processed. This formalism does lend itself to extensions, as we believe that progress on this path will lead us to incorporate additional information sources and components.

In order to illustrate how M2D2 could be used, we also presented three examples of correlation (actually of aggregation) methods. These examples are rigorously specified using the formal definition of M2D2. As opposed to already published correlation methods, these examples use more than the events generated by security tools; they make use of many concepts formalized in M2D2.

References

1. G. Jakobson and M. D. Weissman. Alarm correlation. *IEEE Network Magazine*, pages 52–60, 1993.
2. J. McHugh. Intrusion and intrusion detection. *International Journal of Information Security*, July 2001.
3. Icat vulnerabilities database. http://icat.nist.gov/icat.cfm.
4. G. Vigna and R. A. Kemmerer. Netstat: A network-based intrusion detection approach. In *Proceedings of the 14th Annual Computer Security Application Conference*, December 1998.
5. G. Vigna and R. A. Kemmerer. Netstat: A network-based intrusion detection system. *Journal of Computer Security*, February 1999.
6. R. P. Goldman, W. Heimerdinger, S. A. Harp, C. W. Geib, V. Thomas, and R. L. Carter. Information modeling for intrusion report aggregation. In *Proceedings of the DARPA Information Survivability Conference and Exposition*, June 2001.
7. G. Vigna. A topological characterization of tcp/ip security. Technical Report TR-96.156, Politecnico di Milano, 1996.
8. J.-R. Abrial. *The B Book: Assigning programs to meanings*. Cambridge University Press, 1996.

9. R. Shirey. Internet security glossary. RFC2828, 2000.
10. J. Arlat, J.P. Blanquart, A. Costes, Y. Crouzet, Y. Deswarte, J.C. Fabre, H. Guillermain, M. Kaaniche, K.Kanoun, J.C. Laprie, C. Mazet, D. Powell, C. Rabejac, and P. Thévenod. *Guide de la sureté de fonctionnement*. Cepadues editions, 1995.
11. D. E. Mann and S. M. Christey. Towards a common enumeration of vulnerabilities. In *Proceedings of the 2nd Workshop on Research with Security Vulnerability Databases*, January 1999.
12. Dave Curry and Hervé Debar. Intrusion detection message exchange format data model and extensible markup language (xml) document type definition. Internet Draft (work in progress), December 2001. http://search.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-06.txt.
13. Hervé Debar and Andreas Wespi. Aggregation and correlation of intrusion-detection alerts. In Wenke Lee, Ludovic Mé, and Andreas Wespi, editors, *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection (RAID 2001)*, number 2212 in Lecture Notes in Computer Science, pages 85–103, Davis, CA, USA, October 2001. Springer.
14. J. D. Howard and T. A. Longstaff. A common language for computer security incidents. CERT - SAND98-8667, http://www.cert.org/research/taxonomy_988667.pdf, 1998.
15. F. Cuppens. Managing alerts in multi-intrusion detection environment. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC'01)*, 2001.
16. F. Cuppens and A. Miege. Alert correlation in a cooperative intrusion detection framework. In *Proccedings of the IEEE Symposium on Security and Privacy*, 2002.
17. Frédéric Cuppens and Rodolphe Ortalo. Lambda: A language to model a database for detection of attacks. In H. Debar, L. Mé, and S. F. Wu, editors, *Proceedings of the Third International Workshop on the Recent Advances in Intrusion Detection (RAID'2000)*, number 1907 in LNCS, pages 197–216, October 2000.