ONTIDS: A flexible context-aware and ontology-based alert correlation framework

Alireza Sadighian¹, José M. Fernandez¹, Antoine Lemay¹, and Saman T. Zargar²

¹ Department of Computer & Software Engineering
École Polytechnique de Montréal, Montréal, Canada
{alireza.sadighian, antoine.lemay, jose.fernandez}@polymtl.ca

² School of Information Sciences, University of Pittsburgh, Pittsburgh, PA, USA,
stzargar@sis.pitt.edu

Abstract. In order to reduce the numbers of non-relevant alerts and false positives typically generated by Intrusion Detection Systems (IDS) in real-world situations, several alert correlation approaches that integrate and jointly analyse the alert streams of different alert sensors have been proposed. Inspired by the mental process of contextualisation used by security analysts to weed out less relevant alerts, some of these approaches have tried to incorporate into the correlation process contextual information such as type of systems, applications, users, and networks. However, they tend to be limited in flexibility, as they only perform correlation based on narrowly defined definitions of context. In order to provide a method to automate the analysis of the various information resources available to the security analyst, while preserving maximum flexibility and power of abstraction in the definition and use of such concepts, we propose the ONTIDS ontology-based correlation alert framework. ONTIDS uses ontologies to represent and store information on alerts, context and vulnerability information, and attack scenarios, and uses simple ontology logic rules written in Semantic Query-Enhance Web Rule Language (SQWRL) to correlate and filter out non-relevant alerts. We illustrate the potential usefulness and flexibility of our framework by describing a reference implementation that we use on two separate analysis case study scenarios, inspired from the DARPA 2000 and UNB ISCX IDS evaluation datasets.

Keywords: Intrusion detection, alert correlation, ontology.

1 Introduction

Intrusion detection systems (IDS) collect data from the IT infrastructure and analyse it to try to identify ongoing attacks. Various IDS types have been proposed in the past two decades and commercial off-the-shelf (COTS) IDS products have found their way into Security Operations Centres (SOC) of many large organisations. Nonetheless, the usefulness of single-source IDS has remained relatively limited due to two main factors: their inability to detect new types of
attacks (for which new detection rules or training data are unavailable) and the often very high rate of false positives.

One of approaches that has been suggested to address these problems is that of *alert correlation*, where the alert stream from several different IDS, or more generally various alert sensors, is jointly considered and analysed to provide a more accurate threat picture. When each of these IDS examine the same type of data, one can speak of *homogeneous IDS correlation*. In fact, the majority of research and real-world deployment of correlation approaches involves the analysis of alerts generated by different network IDS (NIDS), such as SNORT or Bro, examining network traffic streams at different network locations. One notorious sub-case of homogeneous correlation is *alert fusion*, where all IDS are examining events from the *exact same* data source and where a decision as to which alert-generating events are most relevant.

Nonetheless, most attacks, whether automated malware infections or manual network intrusions, do not leave traces only on network traffic captures but also on host-based IDS (HIDS) and other security products, and sometimes even on non security-related logs of commodity or corporate applications. This fact has been successfully exploited by security analysts worldwide to detect sophisticated attacks by visually or manually correlating these various information and alert sources. Because all of these sensors examine different types of events and raw data sources, one can speak in this case of *heterogeneous alert correlation*.

One of the important difficulties of heterogeneous correlation is the integration of data from various alert sources, each having potentially different formats and semantics. In order to be useful, the integrated information must capture the generic properties pertaining to all types of alerts in order to allow the analyst to consider the information as a whole. At the same time, sensor-specific attributes must also be retained in order to preserve the ability for the security analyst to drill down and refine his analysis, such as for finding root causes, determining attack type, objectives, etc. Having recognised the usefulness of alert correlation, whether homogeneous or heterogeneous, security researchers have attempted to create unified models for events and alerts, such as the notorious Intrusion Detection Message Exchange format (IDMEF) [1], which is now supported by many COTS NIDS and HIDS. However, IDMEF does not solve all integration woes. It does not gracefully support non-standard attributes that might be needed for refined analysis (except through “user” fields) and is not suited for integrating other types of information that security analysts might want to correlate, such as application logs, configuration information, etc.

In fact, one very fundamental principle of alert management is that security analysts must be able to understand and consider in which *context* the alert originated, i.e. which type of machine, in what part of the network, what application, etc. This is what allows to consider the relevance and relative importance of alerts. In generic terms, this potentially includes any information that can be used to characterise the situation of an IT entity, where an IT entity can be a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves [2]. Contextual
information that can be relevant to security analysis can include network topology and protocols, network, system or application configurations, user profiles and roles, etc. Unfortunately, security analysts often need to manually gather such information from multiple systems to feed the correlation process in order to integrate and validate the alerts and identify the consequences of any intrusion. This is why certain researchers have proposed approaches to automatically include such contextual information into the alert correlation process, an approach referred to as context-aware alert correlation. The simple false positive-reducing idea applied here is simple and intuitive: alerts that are related to a certain type of attack are only relevant if the context in which they happen is indeed vulnerable to that type of attack. Thus, for context aware alert correlation to be useful it must also consider vulnerability information and, potentially, also attack models that describe how attacks require vulnerabilities and how they generate alert-triggering events.

Here again, the difficulty in implementing such approaches resides in integrating the information into a data model that is generic enough to be allow a global view of the data, while retaining maximum data granularity for drill-down analysis. Furthermore, whether we are considering data representations for alerts, context, vulnerabilities or attacks, the ever-changing nature of threats and of our own IT infrastructures make it unattainable to try to design a unified one-size-fits-all data model. Flexibility and extensibility of the data model is thus a key requirement of any such approach. Lastly, the method by which security analysts extract information and intelligence from such data stores must itself also be flexible and extensible. It must support generic simple queries and detailed analysis, and furthermore it must be relatively simple and quick for analyst to implement and run various correlation paradigms and algorithms.

To this purpose, we present in this paper ONTIDS, an automated and context-aware alert correlation framework that relies heavily on ontologies and ontology description logic to accomplish these goals. ONTIDS has the following characteristics:

1. It performs heterogeneous alert correlation, in order to detect complex attacks that might leave traces in different types of sensors.
2. It includes a set of comprehensive but extensible ontologies, allowing correlation and reasoning with information collected from various resources, including context information, vulnerability databases and assessments, and attack models.
3. It can be used to seamlessly and automatically implement various alert correlation approaches on the same data model.
4. It can be applied in different deployment and analysis contexts, from simple to complex IT infrastructures, generic threat detection to complex attack forensics analysis.

The rest of this paper is organised as follows. In Section 2, we discuss the related work. In Section 3, we present our framework in detail. We demonstrate in Section 4 the flexibility of our framework by describing a reference implementation and applying it to the analysis of two different case studies. We conclude
in Section 5, by describing the limitations of these case studies to conclusively demonstrate reduction of non-relevant alerts and false positives, the potential deployment challenges of our framework in real world scenarios, and finally describing possible research avenues to attempt to address some of these questions.

2 Related Work

In a keynote publication Valeur, Vigna, Krügel and Kemmerer [3] define alert correlation as the multi-step process of gathering alerts from a number of IDS and producing a high-level description of the malicious behaviour. In their work, they propose a correlation approach consisting of ten steps, which we will use later to exemplify our generic framework in Section 3.5. This is perhaps the most comprehensive approach, with other work concentrating only on one particular aspect of the correlation process, such as alert fusion [4, 5] or attack thread reconstruction [6]. In fact, one key element of alert correlation is the attack reconstruction step. From a classification point of view, Cuppens and Miege [7] classify attack reconstruction approaches into two categories:

1. **Explicit alarm correlation**, which relies on the capabilities of security administrators to express logical and temporal relationships between alerts in order to detect complex multi-step attacks. For instance, Morin and Debar [8], propose an explicit correlation scheme based on the formalism of chronicles. Other researchers have proposed imperative languages to express logical and temporal relationships in attacks in order to correlate sequences of the alerts [9,10].

2. **Implicit alarm correlation**, which is based on employing machine learning and data mining techniques to fuse, aggregate and cluster alerts for alert correlation and intrusion detection purposes. For instance, Chen and Aritsugi [11], employ Support Vector Machine (SVM) and co-occurrence matrix in order to propose a masquerade detection method. In [12], Raftopoulos performs log correlation using C4.5 decision tree classifiers after analysing the diagnosis of 200 infections that were detected within a large operational network. Almgren, Lindqvist and Jonsson [13] use Bayesian networks to correlate alerts generated from several audit sources to improve detection accuracy.

One of the shortcomings of the approaches in both categories is that they do not take into account all available and important information resources, such as context and vulnerability information. Contextual information has proved useful in better identifying specific alerts or to improve IDS efficiency. Gagnon, Massicotte and Esfandiari [14] have studied the use of target configuration as context information in order to identify non-critical alerts [14]. The Workload-aware Intrusion Detection (WIND) proposal by Sinha, Jahanian, and Patel [15] combines network workload information with Snort rules to improve its efficiency. Unfortunately, these studies only consider partial contextual information, such as target configuration or network traffic, and do not allow for inclusion of other types of context concepts.
Ontologies are knowledge representation models that allow the description of concepts, their attribute and the inheritance and association relationships between, in a way very similar to object-oriented modelling languages such as the Unified Modelling Language (UML). In addition, various types of ontologies have formal description languages that allow for the definition of complete reasoning logic that are machine-interpretable and solvable. Hence, they can be suited for representing concepts and for automated reasoning on domain-specific applications with a limited number of concepts. For that reason, some researchers have proposed ontology-based alert correlation approaches in alert correlation. Vorobiev [16] proposed security ontologies to improve IDS capabilities for detecting new types of attacks such as multi-step distributed attacks and various distributed denial of service (DDoS) attacks. The proposed ontologies, however, only include general security concepts, and no discussion on how they can be adapted to different contexts. The Intrusion Detection and Diagnosis System (ID2S) proposed by Coppolino, D’Antonio, Elia, and Romano [17] uses ontologies as well to correlate detection information at several architectural levels for further intrusion symptom analysis. Finally, Wang, and Guo [18] have proposed an ontology-based approach to model security vulnerabilities listed in the National Vulnerability Database (NVD) [19].

In summary, while Valeur et al. [3] provides a good generic framework for alert correlation into which the various other attack reconstructions approaches can be incorporated [6,9–13], none of these attacks contrast alert information with context. On the other hand, those alert correlation approaches that do, have limited notions of context that cannot be readily extended, and do not do attack reconstruction. Finally, correlation approaches that have employed ontologies have not taken full advantage of their expressive power in terms of data modelling and logic reasoning.

Motivated by these shortcomings, and in order to provide a common solution encompassing the advantages of all of these approaches, we have designed and propose henceforth the ONTIDS alert correlation framework. In order to address the data integration problems while attaining the flexibility and extensibility objectives mentioned in Section 1, ONTIDS uses ontologies and ontology description logic to integrate alerts, contextual information, vulnerability databases, and attack scenarios (or any other type of relevant information).

3 The ONTIDS alert correlation framework

The ONTIDS framework was made context-aware in order to take full advantage of the context information that security analysts have typically access to prioritise alerts, and ontology-based in order to provide a technological solution to the problem of heterogeneous data integration and retrieval. The ONTIDS framework is depicted in Fig. 1. In its first step, the alerts generated via distributed homogeneous or heterogeneous IDS are collected and transferred into the alert
integration component. Also in this step, all the information required for reasoning on these alerts is gathered from three different information resources namely: Context Sensors (CS), common vulnerability databases, and attack databases.

Fig. 1. The ONTIDS ontology-based context-aware alert correlation framework

The second step consists of the following two tasks: i) integrating and converting all the alerts generated by the various IDS into a unified format analysable by the alert correlation unit, and ii) integrating all the contextual information received implicitly or explicitly from the various tools implemented in the system.

In the third step, the alert and context ontologies are populated based on the integrated and converted alert and context information. In order to fully automate the alert correlation process, we have designed a group of comprehensive and extensible ontologies, namely alert, context, attack and vulnerability ontologies. The explicit relationships between these ontologies reasoning on the information gathered from various resources, including the (mostly) static attack and vulnerability databases.

The last step consists in correlating the existing information within the ontologies, which is done via the correlation engine using ontology description logic. We now describe each of the above component in detail.

3.1 Information resources

Alert sensors generate alerts based on the suspected malicious behaviours that they observe on the systems they monitor. The most typical and commonly deployed type of sensor are NIDS that generate alerts by examining individual network traffic packets. They can also include host-based IDS that generate alerts based on system or application activity observed on a particular machine. Finally, it can also include other type of non security-related sensors such as...
application and system logs that are not generating alerts per se, but rather system events that the security analyst consider important enough to be correlated with other sources of alert. The difficulty here is that while many NIDS and HIDS will generate IDMEF-compliant alerts by filling generic attributes (e.g. time, severity, etc.), there might some sensor- or log-specific attributes that we might want to correlate on, and that must therefore be integrated also. This is what ontologies are particularly suited for.

**Context sensors** is a generic term for any information source that can provide contextual information about the systems that are being monitored. The concept of context is purposefully vague to allow analysts to define and use the particular aspects that they think is suitable for monitoring of their systems. This can include different types of information such as configuration (network, host or application), vulnerabilities, user role and profile, location, and even criticality of the corresponding IT asset. Contextual information can be implicitly collected by methods such as vulnerability scanning, network fingerprinting, passive network monitoring tools, or they can be explicitly provided by system administrators through tools such as Configuration Management Systems (CMS), for example.

**Known vulnerabilities** At first, we gather information about vulnerabilities from the well-known public databases such as the Common Vulnerabilities and Exposures (CVE) [20] or the NVD. Then, vulnerabilities can be associated to context instances (e.g. hosts, networks, applications) through vulnerability scanning or asset management. Severity information from these databases, in combination with information on asset criticality, can then be used to help prioritise alerts occurring in these contexts.

**Attack scenarios and models** Attack information and models can obtained from standardised databases such as the Common Attack Pattern Enumeration and Classification (CAPEC) [21] or expert knowledge. In order to model attacks, any of the existing attack modelling languages such as LAMBDA [9] or STATL [22] could be used. However, it is outside of the scope of this work to implement these formalisms within the ontology description logics that we use. In the rest of this article, and without loss of generality, we will illustrate our framework using a simplified attack model comprising a linear sequence of steps.

### 3.2 Alert and context integration

Different types of IDS sensors produce alerts in various formats that might not be natively interpretable by the correlation unit. Hence, it is necessary to preprocess these alert streams and export them in a format that is understandable by the correlation unit. In production environments, this would be done by sensor-specific drivers that would match alert fields with class attributes at the appropriate abstraction level. In following good ontological engineering practises, all
alert sensor-specific fields should be translated into class attributed at the highest possible class in the taxonomy of alerts, i.e. that where all subclasses contain that type of information (or an equivalent one). The use of standard representations such as IDMEF [1] or the Common Event Expression (CEE) [23] should be encouraged, but not at the detriment of not integrating sensor-specific information that is not standard-compliant; that is what sensor-specific alert subclasses are for. For simplicity of presentation and for illustrative purposes, we use an IDMEF-specific ontology in the rest of this paper.

The context integration component of our framework also integrates all the contextual information in various formats received implicitly or explicitly from various tools implemented in the system. In this component, the contextual information gathered using our designed drivers is converted into a unified format analysable by the other components, i.e. into instances in the context ontology. Once the integration process is complete, the correlation process can start.

### 3.3 Description of the ontologies

We chose to use ontologies because they provide a powerful knowledge representation information structure in a unified format that is understandable by both machines and humans. Ontologies also allow the use of reasoning logic formalisms, that can be used to retrieve information in a generic and class structure-agnostic fashion. In our case, we use these reasoning logic formalisms to design alert correlation algorithms, that will attempt to reconstruct possible attack scenarios while eliminating improbable ones, while making abstraction of irrelevant sensor- or system-specific details. The use of ontologies and ontology description logic thus enables us to fully automate the correlation process that is typically done manually by security analysts, and this uniformly considering all relevant information, no matter what its original format or source.

In order to integrate the data inputs to the correlation process and allow generic correlation reasoning, independent of specificities of information resources, we have constructed basic ontologies capturing the essence of the concepts of alert, context, vulnerability, and attack. Essentially, they correspond to the following intuitive security facts:

1. Attack scenarios will generate system events that might in turn trigger sensors to cause related alerts. Depending on the attack model, an attack scenario might be described as linear sequence of events, or a partial ordering of events with pre- and post-conditions, an attack graph, etc.
2. Alerts happen in a context, whether this is an IT asset, network location, application, user, etc. In our case this relationship will be made explicit through information provided by the sensor with the alert (e.g. IP address).
3. Vulnerabilities are always associated to a context, whether to high-level context concepts (e.g. an asset or service type) or to lower-level context subclasses (e.g. particular versions of OS or applications). Conversely, explicit context instances can be linked to specific or generic vulnerabilities, through vulnerability assessment or CMS information.
4. (Most) attack scenarios will require certain vulnerabilities to be present on the systems (context) so that they can **exploited** by that attack.

Figure 2 illustrates these class relationships and some of the subclasses of the basic ontology. These “starter” ontologies are not meant to be the end state of knowledge representation that security analysts would need in using our framework, but rather a starting point or template from which to build on, depending on the kind of sensors, context information or granularity of vulnerabilities and attack modelling desired. We now describe each of these ontologies in more detail.

**Alert ontology.** All the integrated alerts are transferred into this ontology as its instances. As explained above, it has dependency relationship with the context ontology and an association relationship with the attack ontology, since usually each alert \( a \) is typically by a (suspected) attack \( \alpha \) in a particular context \( c \). The generic base class Alert in Fig. 2 includes generic alert attributes such as source, target, time, and analyser (i.e. sensor). The Alert class has three subclasses NetAlert, HostAlert, and AppAlert, corresponding to alerts generated by NIDS, HIDS, and application-based IDS or application logs, respectively, each containing specific sensor-generated attributes. It is important to note that because the concept of context is potentially very rich and multifaceted, it is likely that a single alert might have to be linked to multiple context instances from various subclasses (e.g. a user, a network segment, an application), and thus the association between alert and context will be many-to-many at the Context base class level.

Context ontology. The integrated contextual information is transferred into the context ontology. We split contextual information into two categories: i)
static context information that rarely changes over time (e.g. network architecture, host/user profiles, and OS type), and ii) dynamic context information that changes continuously over time (e.g. traffic type, system usage, time of day/week). As depicted in Fig. 2, the context ontology includes a Context base class and User, Host, Network and Service subclasses with their corresponding attributes. As mentioned, the implicit and explicit context information appear as instances of the context class in our proposed ontology, that will be populated through static information from CMS or system administrators, dynamic information from network profiling tools and even alert sensors themselves (e.g. when reporting on previously unknown IT assets/contexts).

Vulnerability ontology. This ontology represents the list of vulnerabilities related to the existing assets in the underlying context, typically populated from a public vulnerability source such as CVE or NVD. This ontology has a part-whole relationship (composition) with the context ontology since every vulnerability \( v \) is specific to a particular type of system, which is represented as a subclass of Context (typically Host). Thus, \( v \) can be associated with all the asset (context) instances \([c_1, \ldots, c_n]\) that are vulnerable to it, by querying the ontology for Host instances whose applications (App) or OS are those associated to that vulnerability. This ontology also has an association relationship with attack ontology, since usually every vulnerability \( v \) is exploitable by one or more attacks.

Attack ontology. The attack ontology includes information related to the known attack scenarios, and it includes generic attack attributes such as vectors, objectives, and so on. The Vector class represents the method that is used by an attack to infect computer systems, with subclasses in Fig 2 including social engineering, phishing and removable media are examples of such methods. The Objective class includes subclasses such as information leakage, remote core execution, spamming and privilege escalation. The attack ontology has dependency relationships with the context ontology, and association relationship with the alert and vulnerability ontologies, since basically every attack \( at \) needs a particular context \( c \) to proceed, it might need to exploit particular vulnerabilities \([v_1, \ldots, v_n]\), and it results in triggers some alerts \([a_1, \ldots, a_n]\).

3.4 Correlation engine

In order to implement the correlation logic, we employ Ontology Web Language-Description Logic (OWL-DL) to design and populate an ontology for each of the above four inputs. The use of generic language like OWL-DL provides significant flexibility to the framework by allowing the reuse or adaptation of data queries expressed in that logic to various deployment and security monitoring scenarios, e.g. on-line detection or after-the-fact network forensics analysis.

Generally speaking, the correlation engine will have information about only three of the four ontologies that we have defined, i.e. alert, context and vulnerabilities. It is by following the above-mentioned relationships between the
corresponding classes that the correlation engine will be able to infer whether there is an attack instance that could match a particular subset of linked alerts, contexts and vulnerabilities. This correlation process is two-fold and can be viewed as two independent traversals on the core ontology classes:

1. **Context- and vulnerability-based filtering.** Given an alert (or alerts) determine which contexts instances are involved, what are their associated known vulnerabilities, and finally determine which attack scenarios could be exploiting them.

2. **Attack reconstruction.** For each possible attack scenario related to this (or these) alert(s), try to match the sequence of previous alerts with the steps of the attack.

The outcome of this process should hopefully provide the security analyst with a reduced list of high level descriptions of potential ongoing (or completed) attacks that includes few redundancies, non relevant scenarios and false positives.

In order to implement both components of this alert correlation approach we use a set of logic rules expressed in Semantic Web Rule Language (SWRL) and Semantic Query-Enhanced Web Rule Language (SQWRL). While various specific correlation approaches could be implemented within the above generic model, we use certain aspects of the approach described in [3] to illustrate the use of our framework.

### 3.5 Example implementation of Valeur et al.’s approach

The alert correlation approach proposed in [3], includes a comprehensive set of steps that covers various aspects of the alert correlation process. In order to show how ONTIDS automatically implement these steps, in the following, we explain the implementation details of some of these steps employing ONTIDS.

**Alert fusion.** Alert fusion is the process of merging alerts that represent the independent detection of the same malicious event by different IDS. An important condition in order to fuse two or more alerts is that they should be in the same time window. We have defined Rule 1 within the correlation engine in order to perform alert fusion:

**Rule 1**

\[
\begin{align*}
\text{ALERT}(\text{?a1}) & \land \text{ALERT}(\text{?a2}) & \land \text{ANALYSER}(\text{?an1}) & \land \text{ANALYSER}(\text{?an2}) & \land \text{DetectTime}(\text{?dt1}) \\
\text{DetectTime}(\text{?dt2}) & \land \text{SOURCE}(\text{?s1}) & \land \text{SOURCE}(\text{?s2}) & \land \text{TARGET}(\text{?tar1}) & \land \text{TARGET}(\text{?tar2}) \\
\text{CLASSIFIC}(\text{?cl1}) & \land \text{CLASSIFIC}(\text{?cl2}) & \land \text{ASSESSMENT}(\text{?as1}) & \land \text{ASSESSMENT}(\text{?as2}) \\
\text{stringEqual}(\text{?s1}, \text{?s2}) & \land \text{stringEqual}(\text{?tar1}, \text{?tar2}) & \land \text{stringEqual}(\text{?cl1}, \text{?cl2}) \\
\text{stringEqual}(\text{?as1}, \text{?as2}) & \land \text{subtractTimes}(\text{?td}, \text{?dt1}, \text{?dt2}) & \land \text{lessThan}(\text{?td}, \text{"5s"})
\end{align*}
\]

\[\rightarrow \text{sqwrl:select}(\text{?a1})\]
Alert Verification. Alert verification is the process of recognising and reducing non-relevant alerts which refer to the failed attacks. The major reason of attack failure is the unavailability of the contextual requirements of the attack, i.e. the absence of required vulnerabilities in the attack context. Identifying failed attacks allows the correlation engine to reduce the effects of non-relevant alerts in its decision process. Rules 2 and 3 within the correlation engine of our framework perform alert verification based on the targeted system vulnerabilities:

Rule 2

\[
\text{ALERT(?a) \land HOST(?h) \land OS(?o) \land VULNERABILITY(?v) \land CLASSIFICATION(?cl) \land REFERENCES(?ref) \land hasTarget(?a, ?h) \land hasClassific(?a, ?cl) \land hasOS(?h, ?o) \land hasReference(?c, ?ref) \land hasVulnerability(?o, ?v) \land hasName(?ref, ?n1) \land hasName(?v, ?n2) \land stringEqual(?n1, ?n2) \rightarrow sqwrl:select(?a)
\]

Rule 3

\[
\text{ALERT(?a) \land HOST(?h) \land APP(?ap) \land VULNERABILITY(?v) \land CLASSIFICATION(?cl) \land REFERENCES(?ref) \land hasTarget(?a, ?h) \land hasClassific(?a, ?cl) \land hasApp(?h, ?ap) \land hasReference(?c, ?ref) \land hasVulnerability(?ap, ?v) \land hasName(?ref, ?n1) \land hasName(?v, ?n2) \land stringEqual(?n1, ?n2) \rightarrow sqwrl:select(?a)
\]

Attack thread reconstruction. Thread reconstruction is the process of merging a series of alerts that refer to an attack launched by one attacker against a single target, and is another step in the alert correlation process of [3]. Similarly to the alert fusion process, the alerts should happen in the same time window to be correlated. Rule 4 performs the thread reconstruction process:

Rule 4

\[
\text{ALERT(?a) \land HOST(?h1) \land HOST(?h2) \land TIME(?t1) \land TIME(?t2) \land hasSource(?a, ?h1) \land hasTarget(?a, ?h2) \land hasDetectTime(?a, ?dt) \land greaterThanOrEqual(?dt, ?t1) \land lessThanOrEqual(?dt, ?t2) \rightarrow sqwrl:select(?a, ?h1, ?h2)
\]

In summary, we can see how the first component of our canonical description is implemented by the correlation engine by applying first Rules 1, 2 and 3 to reduce non-relevant alerts. For this purpose, it retrieves required information from the alert, context, and vulnerability ontologies. Next, and for those alerts and scenarios that are relevant attack thread reconstruction is performed by applying Rule 4, where the engine attempts to make a mapping between the filtered alerts and the steps of attacks in the attack ontology. Once it finds any mapping between the two ontologies, it will output the whole attack scenario.
4 IMPLEMENTATION & EVALUATION

In order to illustrate and validate our approach, we constructed an example reference implementation using various ontology representation and reasoning tools, and used it to conduct some simple security analysis on the well-known UNB ISCX and DARPA 2000 datasets. In this section, we start by describing the reference implementation we built to illustrate the framework. We then use it to demonstrate the capabilities and flexibility of framework, by describing how it is used in two distinct case studies.

4.1 Reference Implementation

To illustrate the integration of distinct IDS, we have selected the Snort [24] and IBM RealSecure [25] (a.k.a. ISS RealSecure) NIDS as our alert sensors. As an alert integration tool, we use Prelude [26], which is an agent-less, universal, Security Information Management System (SIM, a.k.a SIEM), released under the terms of the GNU General Public License.

We use the Prot´eg´e ontology editor and knowledge acquisition system to design and implement the ontologies using the Ontology Web Language Description Logic (OWL-DL). We instantiate the above-mentioned ontologies from information coming from the alert integration component, the contextual information gathering sensors such as Nessus [27] and Nmap [28], the CVE vulnerability database, and the designed attack scenarios. Prot´eg´e has a number of plug-ins to transfer data from both XML and relational databases. We use the DataMaster plug-in [29] in order to transfer data from relational databases and the XML Tab plug-in to transfer data from a XML files.

Finally, we utilise the Pellet plug-in [30] as a reasoner for OWL-DL, the Jess rule engine [31] as Semantic Web Rule Language (SWRL) rule compiler, and Semantic Query-Enhanced Web Rule Language (SQWRL) [32] in order to query the ontologies for various purposes.

4.2 Case Study 1: Island-hopping attacks

As our first case study, we describe an instance of island-hopping attack scenario described in [3] which is part of the UNB ISCX Intrusion Detection Evaluation Dataset [33]. As shown Fig. 3, in this scenario the attacker employs the Adobe Reader `util.printf()` buffer overflow vulnerability (CVE-2008-2992) to execute arbitrary code with the same privileges as the user running it.

Let us first follow each of the steps taken by the attacker and let us consider what kind of artifacts would be picked up by alert sensors. To launch the attack, the attacker creates a malicious PDF file using Metasploit (for example), and embeds a Meterpreter reverse TCP shell on port 5555 inside it. Then, the attacker sends a system upgrade email including the PDF file on behalf of admin@[...] to all the users of the testbed. Through user5, who initiates the first session (alert 1), the attacker starts to scan potential hosts on two subnets 192.168.1.0/24 and 192.168.2.0/24 (alert 2). User12 is identified as running Windows XP SP1 with a
vulnerable SMB authentication protocol on port 445 (CVE-2008-4037) (alerts 3 and 4). The attacker exploits this vulnerability to capture user12 (alert 5), and a scan is performed from this user to the server subnet (192.168.5.0/24) (alert 6). This scan identifies a Windows Server 2003 running an internal Web application using MS SQL Server as its backend database with only port 80 opened. This leads to the use of Web application hacking techniques such as SQL injection. Finally, the attacker compromises the target system (alerts 7 and 8). Table 1 presents a summary of the alerts, and indicates their corresponding steps.

Table 1. Alerts generated by alert sensors in the island-hopping attack scenario

<table>
<thead>
<tr>
<th>Alert ID</th>
<th>Name</th>
<th>Sensor</th>
<th>Source</th>
<th>Target</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Local Exploit</td>
<td>HIDS</td>
<td>192.168.1.105</td>
<td>192.168.1.105</td>
<td>Step 1</td>
</tr>
<tr>
<td>2</td>
<td>Scanning</td>
<td>NIDS</td>
<td>192.168.1.0/24</td>
<td>192.168.1.105</td>
<td>Step 2</td>
</tr>
<tr>
<td>3</td>
<td>Windows File Sharing</td>
<td>NIDS</td>
<td>192.168.1.105</td>
<td>192.168.2.112</td>
<td>Step 3</td>
</tr>
<tr>
<td>4</td>
<td>Windows File Sharing</td>
<td>NIDS</td>
<td>192.168.1.105</td>
<td>192.168.2.112</td>
<td>Step 3</td>
</tr>
<tr>
<td>5</td>
<td>Local Exploit</td>
<td>HIDS</td>
<td>192.168.2.112</td>
<td>192.168.2.112</td>
<td>Step 3</td>
</tr>
<tr>
<td>6</td>
<td>Scanning</td>
<td>NIDS</td>
<td>192.168.5.0/24</td>
<td>192.168.5.123</td>
<td>Step 4</td>
</tr>
<tr>
<td>7</td>
<td>HTTPWeb</td>
<td>NIDS</td>
<td>192.168.2.112</td>
<td>192.168.5.123</td>
<td>Step 4</td>
</tr>
<tr>
<td>8</td>
<td>SQLInjection</td>
<td>NIDS</td>
<td>192.168.5.123</td>
<td>192.168.5.123</td>
<td>Step 4</td>
</tr>
</tbody>
</table>
In order to correlate the alerts generated by alert sensors during the above scenario, first, the alert integration component integrates all received alerts. Then, the integrated alerts are transferred into the alert ontology. Additionally, we manually populate vulnerability and context ontologies based on the published documents related to the UNB ISCX dataset. Therefore, the Adobe Reader `util.printf()` vulnerability and others that might be present in the IT infrastructure are input into the vulnerability ontology. Contextual information about the existing hosts (IP addresses, open ports, available services, etc.), services and users are also manually input into the context ontology. In this case, this includes the information about the three compromised hosts (IP addresses 192.168.1.105, 192.168.2.112, and 192.168.5.123), their open ports (i.e., 5555 and 445).

Next, the correlation engine correlates the existing information within the ontologies. For this purpose, we first use the Rules 1, 2 and 3 to eliminate non-relevant alerts. Then, using the following rule we reconstruct the attack scenario.

**Rule 5**

\[
\text{ATTACK}(\text{Pat}) \land \text{hasName}(\text{Pat}, "\text{InsiderAttack1"}) \land \text{ALERT}(\text{a1}) \land \text{CLASSIFICATION}(\text{cl1}) \\
\land \text{HOST}(\text{h1}) \land \text{REFERENCE}(\text{ref1}) \land \text{hasTarget}(\text{a1}, \text{h1}) \land \text{hasClassific}(\text{a1}, \text{cl1}) \\
\land \text{hasReference}(\text{cl1}, "\text{CVE-2008-2992"}) \land \text{ALERT}(\text{a2}) \land \text{hasSource}(\text{a2}, \text{h1}) \\
\land \text{hasName}(\text{a2}, "\text{Scanning"}) \land \text{ALERT}(\text{a3}) \land \text{HOST}(\text{h2}) \land \text{CLASSIFICATION}(\text{cl2}) \\
\land \text{REFERENCE}(\text{ref2}) \land \text{hasTarget}(\text{a2}, \text{h2}) \land \text{hasClassific}(\text{a2}, \text{cl2}) \\
\land \text{hasReference}(\text{cl2}, "\text{CVE-2008-4037"}) \land \text{ALERT}(\text{a4}) \land \text{hasSource}(\text{a4}, \text{h2}) \\
\land \text{hasName}(\text{a4}, "\text{Scanning"}) \land \text{ALERT}(\text{a5}) \land \text{hasSource}(\text{a5}, \text{h2}) \\
\land \text{hasName}(\text{a5}, "\text{SQLInjection"}) \rightarrow \text{SQWL select}(\text{a1}, \text{a2}, \text{a3}, \text{a4}, \text{a5}, \text{at})
\]

Rule 5 correlates alert and attack ontologies, and attempts to discover corresponding alerts for each step of the attack. If it finds at least one match regarding each step, the rule will be successful in detecting the whole attack scenario. Figure 4 represents the result of applying Rule 5 to the ontologies, showing that ONTIDS should be able to reconstruct the attack.

![Fig. 4. The island-hopping attack graph detected by the proposed framework](image)

**4.3 Case Study 2: Recon-breakin-escalate attacks**

As the second case study, we evaluate the proposed alert correlation framework using the DARPA 2000 dataset [34]. The DARPA 2000 dataset contains two
Table 2. Five phases of DARPA’s LLDDOS 1.0 attack scenario

<table>
<thead>
<tr>
<th>Step</th>
<th>Name</th>
<th>Time</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IP Sweep</td>
<td>09:45 to 09:52</td>
<td>The attacker sends ICMP echo-requests and based on the ICMP echo-replies finds out which hosts are active.</td>
</tr>
<tr>
<td>2</td>
<td>Sadmind Ping</td>
<td>10:08 to 10:18</td>
<td>Testing the existence of sadmind daemon on the live IPs.</td>
</tr>
<tr>
<td>3</td>
<td>Exploiting</td>
<td>10:33 to 10:34</td>
<td>Exploiting the sadmind vulnerability to break into vulnerable hosts.</td>
</tr>
<tr>
<td>4</td>
<td>Installation</td>
<td>10:50</td>
<td>Installation of the trojan mstream DDoS software on three hosts.</td>
</tr>
<tr>
<td>5</td>
<td>Launching DDoS</td>
<td>11:27</td>
<td>Launching the DDoS attack.</td>
</tr>
</tbody>
</table>

attack scenarios, LLDDOS 1.0 and LLDDOS 2.0.2, and we have chosen the former for our evaluation. LLDDOS 1.0 is a multi-step scenario corresponding to a Distributed Denial of Service (DDoS) flooding attack. The attack has 5 phases and it takes about three hours to be completed. Table 2 lists the attack phases.

We again use both the RealSecure and Snort NIDS as base our alerts sensors to detect all the steps of the attack. Snort outputs around 1,211 raw alerts for the LLDDOS 1.0 dataset, but it does not detect the installation phase of the DDoS attack (i.e. phase 4). On the other hand, and as is described in [35], RealSecure outputs 924 raw alerts for the same dataset, corresponding to the 22 alert types shown in Table 3. However, it does not output any alerts related to ICMP pings (i.e. phase 1). Consequently, the combination of Snort and RealSecure can detect all phases of the attack. Nonetheless, just using a combination of both IDS alerts with a simple OR rule will result in a significant number of redundant alerts and false positives, as we will see. With ONTIDS, we expect to have lower redundancy, and fewer non-relevant alerts and false positives.

Table 3. Alert types generated by ISS RealSecure based on the DARPA 2000 dataset

<table>
<thead>
<tr>
<th>ID</th>
<th>AlertType</th>
<th>ID</th>
<th>AlertType</th>
<th>ID</th>
<th>AlertType</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RIPExpire</td>
<td>9</td>
<td>Admind</td>
<td>17</td>
<td>SSH_Detected</td>
</tr>
<tr>
<td>2</td>
<td>RIPAdd</td>
<td>10</td>
<td>Sadmind_Ping</td>
<td>18</td>
<td>Email_Debug</td>
</tr>
<tr>
<td>3</td>
<td>Email_Ehlo</td>
<td>11</td>
<td>Email_Amald_Overflow</td>
<td>19</td>
<td>TelnetDisplay</td>
</tr>
<tr>
<td>4</td>
<td>Telnet_Terminaltype</td>
<td>12</td>
<td>HTTP_Java</td>
<td>20</td>
<td>Telnet_EnvAll</td>
</tr>
<tr>
<td>5</td>
<td>FTP_User</td>
<td>13</td>
<td>Sadmind_Amslverify_Overflow</td>
<td>21</td>
<td>Port_Scan</td>
</tr>
<tr>
<td>6</td>
<td>FTP_Pass</td>
<td>14</td>
<td>Mstream_Zombie</td>
<td>22</td>
<td>Stream_DoS</td>
</tr>
<tr>
<td>7</td>
<td>Rsh</td>
<td>15</td>
<td>Telnet_Ssh</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>HTTP-Shells</td>
<td>16</td>
<td>HTTP-iso95_falstgr_exec</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the second step, Prelude converts all the received alerts into the IDMEF format, and transfers the integrated alerts into the alert ontology as its instances. We manually populate the context and vulnerability ontologies based on the information existing in the published documents related to the DARPA 2000 dataset. Thus, the Solaris sadmind vulnerability (CVE-1999-0977) and others existing vulnerabilities in the underlying network are transferred into the vulnerability ontology. The same is done with contextual information about the existing hosts and users, in this case including the three compromised hosts (IP
addresses 172.16.115.20, 172.16.112.50, 172.16.112.10), and their open ports (i.e. telnet port 23) and users (e.g. hacker2).

As before, the correlation engine uses Rules 1–3, to eliminate redundant and non-relevant alerts. Based on our analysis, 32.7% of all alerts were generated by both Snort and ISS RealSecure. In our case, we report these alerts as a single alert (in order to reduce redundancy) since both IDS agree. Alerts reported by only one IDS are then further analyzed by attempting attack reconstruction on the 5 phases of the LLDDOS 1.0 attack scenario, by using the following rule:

**Rule 6**

\[
\text{ATTACK(?at)} \land \text{hasName(?at, "LLDDOS1")} \land \text{ALERT(?a1)} \land \text{ALERT(?a2)} \land \text{ALERT(?a3)} \land \text{ALERT(?a4)} \land \text{ALERT(?a5)} \land \text{ALERT(?a6)} \land \text{ALERT(?a7)} \land \text{HOST(?h1)} \land \\
\text{hasName(?a1, "Scanning")} \land \text{hasTarget(?a1, ?h1)} \land \text{hasService(?h1, "ICMP")} \land \\
\text{hasName(?a2, "Sadmind_Ping")} \land \text{hasName(?a3, "Sadmind_Amslverify_Overflow")} \land \\
\text{hasName(?a4, "Admind")} \land \text{hasName(?a4, "Rsh")} \land \text{hasName(?a4, "MStream_Zombie")} \land \\
\text{hasName(?a4, "Stream_DOS")} \rightarrow \text{sqowl:select(?a1, ?a2, ?a3, ?a4, ?a5, ?a6, ?a7, ?at)}
\]

Rule 6 correlates alert and attack ontologies, and discovers corresponding alerts for each step of the attack. If at least one match is found for each step, the rule will be successful in detecting the whole attack scenario. According to this rule, our results indicate that 91.08% of the alerts were false positives, and only 8.92% of the alerts were true positives.

Table 4 summarises our results. Since both Snort and ISS RealSecure only detect a few of the 33,787 attack events in Phase 5 (launching DDoS), their total false negative rates are quite high. The recall column consequently reports low values for both sensors and ONTIDS. On the other hand, ONTIDS does considerably well at reducing false positives, in fact reducing it to 0.

<table>
<thead>
<tr>
<th>IDS</th>
<th>Redundant alerts (%)</th>
<th>FP</th>
<th>TP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Snort</td>
<td>0.00%</td>
<td>118</td>
<td>93</td>
<td>33814</td>
<td>0.07</td>
<td>2 × 10⁻³</td>
<td>2 × 10⁻³</td>
</tr>
<tr>
<td>RealSecure</td>
<td>0.00%</td>
<td>870</td>
<td>54</td>
<td>33853</td>
<td>0.05</td>
<td>10⁻³</td>
<td>10⁻³</td>
</tr>
<tr>
<td>ONTIDS</td>
<td>32.7%</td>
<td>0</td>
<td>123</td>
<td>33784</td>
<td>1.00</td>
<td>3 × 10⁻³</td>
<td>5 × 10⁻³</td>
</tr>
</tbody>
</table>

4.4 Discussion on flexibility

We have demonstrated the use of the ONTIDS alert correlation framework in two quite different case studies involving considerably distinct attack scenarios. More important than the reduction in false positives (in this somewhat contrived evaluation scenario), the point of this exercise was to show the level of flexibility of such an approach. The fact that the same correlation Rules 1–3 are used for the
context-based alert filtering in both scenarios deceptively hides the fact that the vulnerability and context instances in both cases are quite different as they come from different sources, and hence have different attributes and properties. In fact, Fig. 5 illustrates that the ontologies for both attack scenarios are different as they involve different subclasses of context, sensors, attacks and alerts. Nonetheless, it is the power of abstraction of ontologies (implemented through the construction of adequate information integration drivers) is what allows us to design and use such generic correlation rules. As security analysts start to use ONTIDS, we expect that these ontologies will naturally expand to include new subclasses capturing the idiosyncrasies of the systems being monitored, the various types of sensors monitoring them, and richer and more complex attack models and vulnerabilities. The base ontology and basic correlation of ONTIDS described here should help make a good start of it.

Fig. 5. The involved classes of the designed ontologies on the proposed case studies

5 CONCLUSIONS

In this paper, we introduced ONTIDS an ontology-based automated alert correlation framework to try to benefit from the combined advantages of previous alert correlation approaches (including context awareness), while providing a level of flexibility that would allow it to be used in the many different deployment scenarios that security analysts are likely to face.

The main idea behind ONTIDS is to use and leverage a template ontology containing base classes and some subclasses for the concepts of IT asset context, alert, vulnerability and attack. These ontologies are then populated by instances either automatically through source-specific drivers (such as for IDMEF-compliant alert sensors), or manually for static information (such as context, vulnerability and attack information). The correlation engine is then implemented using logic rules written in Semantic Web Rule Language (SWRL) and Semantic Query-Enhanced Web Rule Language (SQWRL) based on the OWL description logic (OWL-DL). The ontologies and correlation rules described here are generic
enough to i) implement as special cases other existing correlation approaches such as that of Valeur et al., and, ii) be applied with minimal changes to different analysis scenarios, such as in the two case studies demonstrated.

Unfortunately, the examples described in this paper provide only limited validation on the viability and efficiency of our approach in terms of non-relevant alert and false positive reduction. In order to illustrate its use we have chosen to use existing and known datasets, yet these datasets include limited information on context, which we had to artificially insert manually. In addition, they can only be used to generate NIDS alerts, since they contain no host activity information. In these circumstances, it is very hard to exemplify and measure the potential positive real-world impact of correlating across heterogeneous types of alert sensors and system logs, in contrast with existing approaches.

Furthermore, while the main inspiration in creating ONTIDS was to provide security analysts with a flexible tool to conduct analysis of alerts, it remains unclear how well received and easy-to-use in practice such an approach would be. In particular, one might ask the question of how much more difficult it would for them to write C-style NIDS detection rules vs. writing correlation algorithms as rules in SWRL/SQWRL. Ideally, the effort required to learn and use these formalisms (and to master the corresponding ontologies) would ideally have to be evaluated in a field study with real security analysts.

Lastly, we have not addressed at all the issue of scalability and performance of our approach. While ontologies are quite flexible and readily provide the benefits of abstraction, they are not always efficient at updating and quickly providing access to stored data. In the case of alert correlation systems of large IT infrastructures, the vast amounts of data involved are likely to make typical XML flat file or relational database storage unwieldy and inefficient for quick on-line alert correlation. While some specific data storage solutions such as object-oriented databases might help alleviate these problems, significant engineering challenges would have to be solved to make ONTIDS perform at the same line speeds as some current commercial-grade NIDS. Nonetheless, and even if future research does not readily solve these problems, we believe in the eventual applicability and usefulness of ontology-based approaches such as this, in situations where immediate online processing is not a key requirement, such as in network forensics in incident handling situations.

In conclusion, we hope to continue to develop the ONTIDS approach and to evaluate its viability and usefulness by conducting field studies with data collected from real-world systems and analysed by real security analysts. On the one hand, this will force us to test the flexibility of the framework by incorporating richer context and sensor ontologies (possibly stretching the limits of abstraction), while also having to express richer correlation algorithms, possibly based on more sophisticated attack models and description languages such as LAMBDA or STATL. On the other hand, this will also provide us with opportunities to better understand and address some of the deployment challenges associated with ontologies, such as difficulty of use, scalability and performance.
Acknowledgements

This research was sponsored in part by the Inter-networked Systems Security Strategic Research Network (ISSNet), funded by Canada’s Natural Sciences and Engineering Research Council (NSERC).

References

ONTIDS: context-aware, ontology-based alert correlation framework

20. CVE: Common vulnerabilities exposures (CVE), the key to information sharing. http://cve.mitre.org/
23. Mitre Corporation: A standardized common event expression (CEE) for event interoperability
34. MIT Lincoln Laboratory: 2000 DARPA intrusion detection scenario specific data sets (2000)