A semantic web approach to share alerts among Security Information Management Systems

Jorge E. López de Vergara[1], Víctor A. Villagrá[2], Pilar Holgado[1], Elena de Frutos[2], Iván Sanz[3]

[1] Computer Science Department, Universidad Autónoma de Madrid, Calle Francisco Tomás y Valiente, 11, 28049 Madrid, Spain
[2] Telematic Systems Engineering Department, Universidad Politécnica de Madrid, Avenida Complutense, s/n, 28040 Madrid, Spain
[3] Telefónica Investigación y Desarrollo Calle Emilio Vargas, 6 28043 Madrid, Spain
jorge.lopez_vergara@uam.es, villagra@dit.upm.es, mpilar.holgado@estudiante.uam.es, e_le_na@hotmail.com, isahe@tid.es

Abstract. This paper presents a semantic web-based architecture to share alerts among Security Information Management Systems (SIMS). Such architecture is useful if two or more SIMS from different domains need to know information about alerts happening in the other domains, which is useful for an early response to network incidents. For this, an ontology has been defined to describe the knowledge base of each SIMS that contains the security alerts. These knowledge bases can be queried from other SIMS, using standard semantic web protocols. Two modules have been implemented: one to insert the new security alerts in the knowledge base, and another one to query such knowledge bases. The performance of both modules has been evaluated, providing some results.

Key words. SIMS, Semantic Web, IDMEF, SPARQL, Jena, Joseki, RDF, OWL.

1 Introduction

Security is an important issue for Internet Service Providers (ISP). They have to keep their systems safe from external attacks to maintain the service levels they provide to costumers. Security threats are identified at routers, firewalls, intrusion detection systems, etc. generating several alerts in different formats. To deal with all these incidents, ISPs usually have a Security Information Management System (SIMS) [1], which collects the event data from their network devices to manage and correlate the information about any incident. A SIMS is useful to detect intrusions at a global level, centralizing the alarms from several security devices.

A step forward in this type of systems would be the distribution of alerts among SIMS from different ISPs and different vendors for an early response to network incidents. Thus, mechanisms to communicate security notifications and actions have to be developed. These mechanisms will let the collaboration among SIMS to share information about incoming attacks. For this, it is important to homogenise the information the SIMS are going to share. A data model has to be defined to address several problems associated with representing intrusion detection alert data: alert information is inherently heterogeneous, some alerts are defined with very little information and others provide much more information; and intrusion detection environments are different, the same attack can contain different information. Current solutions provide a common XML format to represent alerts, named IDMEF (Intrusion Detection Message Exchange Format) [2]. Although this format is intended to exchange messages, it is not a good solution in a collaborative SIMS scenario, as each SIMS would flood the other SIMS with such messages. It would be better that a SIMS asks other SIMS about certain alerts, and later infers what is its situation based on that information. However, IDMEF has not been defined to query for an alert set.

A way to solve this is to use ontologies [3], which have been precisely defined to share knowledge. Ontologies have been previously proposed to formally describe and detect complex network attacks [4, 5, 6]. In this paper we propose to define an ontology based on IDMEF, where the alerts are represented as instances of Alert classes in that ontology. The use of an ontology language also improves the information definition, as restrictions can be specified beyond data-types (for instance, cardinality). With this ontology, each SIMS can store a knowledge base of alerts, and share it using semantic web interfaces. Then, other SIMS can ask about alerts by querying such knowledge bases through semantic web interfaces. As a result, a SIMS would be able to share their knowledge with other domain SIMS. The knowledge would include policies, incidents, actualizations, etc. In a first phase, this sharing has been constrained to share alert incidents.

The rest of the paper is structured as follows. Next section presents the architecture of collaborative SIMs based on knowledge sharing. Then, IDMEF ontology is explained, showing the process followed in its definition, as well as how to query it. After this, an implementation of the system that receives IDMEF alerts and stores them in a knowledge base is described. Results obtained in the different modules are also provided. Finally, some conclusions and future work lines are given.

2 Semantic collaborative SIMS architecture

The architecture we propose to share information among SIMS is based on semantic web technologies, as shown in **Fig. 1**. This figure represents two SIMS but it can be generalized to several of them. Each SIMS will contain an alert knowledge base that contains instances of the IDMEF ontology, described in next section. Each knowledge base can be queried by other SIMS using a semantic web interface that accepts queries about the ontology.

To implement the web service interfaces in this architecture, Joseki server [7] has been used, based on Jena libraries [8]. Joseki is an HTTP server that implements a

query interface for SPARQL (SPARQL Protocol and RDF Query Language) [9]. Joseki provides a way to deal with RDF (Resource Description Framework) and OWL (Web Ontology Language) data in files and databases. Jena libraries have also been used for both the instance generator and the query generator, using the SDB library [10] to store the ontology in a database backend. Section 4 provides a deep explanation about how they have been implemented.

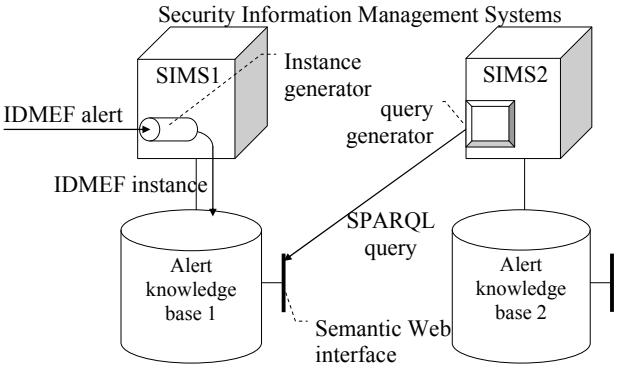

Fig. 1. Semantic collaborative SIMS architecture.

3 IDMEF ontology

IDMEF format provides a common language to generate alerts about suspicious events, which let several systems collaborate in the detection of attacks, or in the treatment of the stored alerts. Although IDMEF has some advantages (integration of several sources, use of a well supported format), it has also drawbacks (heterogeneous data sources led several alerts of a same attack which do not contain the same information).

To solve the identified problems, we have defined an alert ontology based on the IDMEF structure. In this process it is worth remarking that IDMEF has been defined following a model of classes and properties, making easier the ontology definition, with a more or less direct mapping. The ontology has been defined using OWL [11], leveraging the advantages of the semantic web (distribution, querying, inferencing, etc.), and also the results of [12]. Several class restrictions have been defined (cardinality, data types) by analyzing the IDMEF definition contained in [2].

The following conventions have been taken to define the IDMEF ontology:

- Class names start with a capital letter and it is the same as the IDMEF class name.
- Property names starts with a lower-case letter and has the format *domain_propertyName*, where *domain* is the name of the class to which the property belongs, and *propertyName* is the name of the property.

The following rules have also been taken:

- Each class in an IDMEF message maps to a class in the IDMEF ontology.

- Each attribute of an IDMEF class is mapped to a data-type property in the corresponding ontology class.
- Classes that are contained in other class are mapped in general to object-type properties. An exception to this are aggregated classes that contain text, which have been mapped to data-type properties.
- A subclass of an IDMEF class is also represented as a subclass in the ontology, inheriting all the properties of its parent class.
- When an IDMEF attribute cannot contain several values, it is mapped to a functional class.
- When an IDMEF attribute can only have some specific values, the ontology define them as the allowed values.
- Numeric attributes are represented as numeric data-types properties, dates are represented as *datetime* data-type properties, and the rest as *string* data-type properties.

Following the rules above, the ontology has been defined. **Fig. 2** shows a representation of the *Alert* class, its child classes (*OverflowAlert*, *ToolAlert* and *CorrelationAlert*), and other referred classes (*Classification*, *AdditionalData*, *Target*, *Source*, *Assessment*, *CreateTime*, *AnalyzerTime*, *DetectTime*, *Analyzer*). This figure has been generated using the Protégé [13] ontology editor. The boxes represent the classes and the arcs can be inheritance (in black, labelled *isa*) and aggregation (in blue, labelled with the property names) relationships. A UML (Unified Modelling Language) representation could also be provided, using the UML profile for OWL [14].

Our definition enables a mapping from IDMEF messages to IDMEF ontology instances. In this way, the information contained on each IDMEF message is translated to an instance of *Alert*, with instances of *Target*, *Source*, etc. as this information is contained on each message. The ontology includes other additional classes, so any IDMEF message can be represented in the ontology.

With respect to a plain XML IDMEF message, the ontology provides several advantages. For instance, the information can be restricted as defined in the IDMEF definition [2]. Moreover, query languages such as SPARQL can be used to query all the information contained in the knowledge base, and it is not limited to the scope of a concrete XML document, which would be the case of IDMEF messages.

To query the knowledge base, SPARQL has been chosen, given that is has been recently recommended by the W3C as the RDF/RDFS and OWL query language [9]. Using such language a query can be defined as follows:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX idmef: <http://www.dit.upm.es/IdmefOntology.owl#>
SELECT ?alert ?id ?target_address
WHERE {
          ?alert   rdf:type idmef:Alert ;
                   idmef:alert_messageid ?id ;
                   idmef:alert_target ?target .
          ?target idmef:target_node ?tnode .
          ?tnode idmef:node_address ?taddress .
          ?taddress idmef:address_address ?target_address
}
```

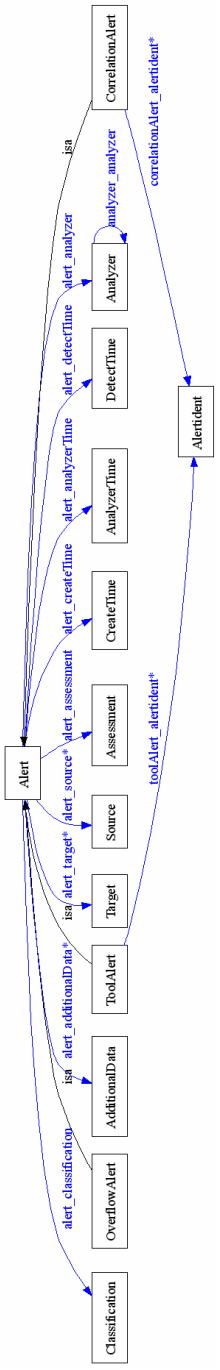

Fig. 2. IDMEF ontology definition.

The query starts with PREFIX clauses, to define the namespaces to be used to identify the queried classes and properties. After this, the variables *alert*, *id* and *target_address* that meet a set of conditions are requested: *alert* variable is of type *Alert*, which has the properties *alert_messageid* and *alert_target*. Then, *alert_target* property refers to an instance with an address value, identified with the variable *target_address*.

4 Implementation

The architecture proposed in section 2 has been implemented. Apart from the components provided by existing semantic web implementations (mainly Joseki server), we have implemented the module that stores the IDMEF alerts in the knowledge base (instance generator), as well as the module that queries alerts of an external knowledge base (query generator). Subsections below present such implementations, providing later some results in section 5.

4.1 Instance generator

A module has been developed to map the IDMEF messages to ontology instances. This module has been developed in Java, taking advantage of the libraries that this language provides for parsing XML documents and ontologies. **Fig. 3** shows the steps that have to be performed to generate and save instances in the knowledge base:

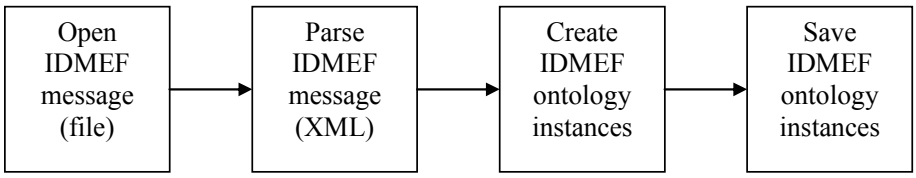

Fig. 3. Steps to generate and store ontology instances.

1. The first step is to open the IDMEF message, contained in a file.
2. Next, the IDMEF message, formatted in XML, is parsed. This generates a tree in memory representing the message. This tree is generated using the SAX Java API. To reduce parsing times, we have let the file to contain several messages. With this approach, we can continuously parse several alerts without needing to restart the process.
3. Then, reading the generated tree, the set of instances of the IDMEF ontology are generated, using the Jena library.
4. Once the instances have been generated, they are saved in a persistent storage, which can be either an OWL file or preferably, a database.

Jena libraries, developed at HP Labs, help when dealing with ontologies in Java applications. In our development we have used Jena version 2, which supports both RDF and OWL languages, as well as a certain level of reasoning on the defined

model. Jena library enables the management of ontologies, adding, deleting or editing tuples, storing the ontologies and querying them. For this, Jena provides classes such as:

- Resource: anything that can be described in a model. Literal is a type of resource that represents a simple data-type, usually a *string*.
- Property: they are characteristics, attributes or relationships used to describe a resource.
- Sentence: A resource joint with a property and an associated value.
- Model: they are set of sentences. They include methods to:
  - Create models.
  - Read and write models.
  - Load models in memory.
  - Query a model: look for information inside the model.
  - Operations on models: union, intersection, difference.

Models can be stored in many ways, including OWL files, as well as representations of the ontology on a relational database. In this last case, there are several storing possibilities, depending on the library used to represent the ontology on the database. Precisely, SDB is a Jena library specifically designed to provide storage in SQL databases, both proprietary and open source. This storage can be done through the SDB API.

4.2 Query generator

The Knowledge base, where the alerts are stored, can be queried through semantic web interface by other SIMS. For this, another module has been developed, which performs SPARQL queries to a Joseki server through HTTP. This server accesses the Knowledge Base and it obtains the results of that query. These results are then received by the query module.

To connect the query module to Joseki, it is necessary to use the ARQ library [15], which is a query engine for Jena. The query module can execute any SPARQL query. For most habitual queries, we have implemented a program which does the query depending on a series of parameters. For instance:

- All alerts depending on the time:
  - Alerts in the last week.
  - Alerts in the current day.
  - Alerts in a day.
  - Alerts in an interval of time.
- Alerts queried using other parameters:
  - Source IP address.
  - Target IP address.
  - Source port.
  - Target port.
  - Alert type.
  - Target of the attack.
  - Source of the attack.

  - Tools of the attack.
  - Overflow Alert.
  - Analyzer.
- Assessments of the attacks: impact, actions, etc.

5 Results

The implemented modules, presented above, have been tested to know their performance. All the results have been obtained in a computer equipped with an Intel Core2 Duo E8500 processor at 3.16 GHz with 6 MB L2 Cache and 2 Gbyte RAM. Previous tests with older computers provided worse results.

5.1 Instance generator

To evaluate the generation of instances, IDMEF messages available in [2] have been used. **Table 1** shows the times measured in milliseconds.

Table 1. Time to generate instances of well known IDMEF messages

IDMEF message	JDBC	SDB	SPARQL/Update
Assessment	1235	1040	-
Correlated Alert	1250	1035	640
Disallowed Service	1250	1050	640
Load Module	1220	1050	625
Load Module 2	1250	1035	640
Phf	1220	1035	610
Ping of Death	1220	1035	625
Policy Violation	1265	1035	640
Scanning	1235	1035	610
Teardrop	1220	1035	610

These times are measured after the database is created and the ontology model is represented on the database. If the database and the model have to be created, there are two possibilities:

- Use of JDBC (Java Database Connectivity), with a time of around 1.900 s.
- Use of SDB library, with a time of around 1.125 s, faster than the previous case.

Both JDBC and SDB libraries facilitate the connection to databases containing ontologies from Java application independently of the operating system. These libraries are also compatible with different databases. In addition, SDB is a Jena component designed specifically to support SPARQL queries and it provides storage in both proprietary and open source SQL databases.

Once the database has been created, there are three alternatives to insert the instances on the ontology database: JDBC, SDB and SPARQL/Update [16]. With respect to the last alternative, SPARQL/Update is an extension to SPARQL that lets a

programmer the definition of insert clauses, whereas JDBC and SDB can insert data in the ontology by creating ontology data structures in memory that are later stored. From our experiments, the best measurements are obtained if the language SPARQL/Update is used to insert the instances. They are approximately a 60% of the time when SDB library is used, and a 50% compared to when plain JDBC is used. In the case of the Assessment message there is an exception, because it contains characters that cannot be used in the SPARQL/Update sentence. In this case, the SDB library should be used instead.

5.2 Query generator

Some measurements have also been taken with respect to the time that it takes to perform a concrete query from the query module to a test knowledge base with 112 alerts through the Joseki server. Simplified versions of the queries used for the experiment are shown below (they also included other variables that could be useful about other alert properties):

- Alerts depending on a time interval:

```
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX idmef: <http://www.dit.upm.es/IdmefOntology.owl#>
SELECT ?alert ?time
WHERE {
          ?alert rdf:type idmef:Alert .
          ?alert idmef:alert_createTime ?createTime .
          ?createTime idmef:createTime_time ?time .
            FILTER (?time > time1).
            FILTER (?time < time2)
}
```

  where *time1* and *time2* are properly replaced to query for a concrete period of time.

- Alerts depending on the source IP address.

```
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX idmef: <http://www.dit.upm.es/IdmefOntology.owl#>
SELECT ?alert ?sourceAddress
WHERE {
          ?alert rdf:type idmef:Alert.
          ?alert idmef:alert_source ?source.
          ?source idmef:source_node ?node.
          ?node idmef:node_address ?address.
          ?address idmef:address_address ?sourceAddress.
          FILTER (?sourceAddress = ipAddr)
}
```

  where *ipAddr* is replaced with a concrete IP address

- Alerts depending on the target IP address.

```
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX idmef: <http://www.dit.upm.es/IdmefOntology.owl#>
SELECT ?alert ?sourceAddress
WHERE {
          ?alert rdf:type idmef:Alert.
          ?alert idmef:alert_target ?target.
          ?target idmef:target _node ?node.
          ?node idmef:node_address ?address.
          ?address idmef:address_address ?targetAddress.
          FILTER (?targetAddress = ipAddr)
}
```

  where *ipAddr* is replaced with a concrete IP address.

- Alerts depending on their type:

```
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX idmef: <http://www.dit.upm.es/IdmefOntology.owl#>
SELECT ?alert ?alertName
WHERE {
          ?alert rdf:type idmef:Alert.
          ?alert idmef:alert_classification ?classification.
          ?classification idmef:classification_text ?aName.
          FILTER (?aName = alertName )
}
```

  where *alertName* is replaced for a concrete alert.

Tables 2, 3, 4 and 5 show below the results obtained when querying the alert knowledge base with these queries:

Table 2. Knowledge base query times depending on the time interval.

Obtained results	Time (ms)
23	547
9	500
32	641

Table 3. Knowledge base query times depending on the source IP of an alert.

Obtained results	Time (ms)
1	453

Table 4. Knowledge base query times depending on the target IP of the alerts.

Obtained results	Time (ms)
11	500
33	625
77	750

Table 5. Knowledge base query times depending on the alert type.

Obtained results	Time (ms)
2	468
13	484
7	468

As shown, the time to retrieve the results is dependent on the number of alerts that match the query, but not on the query itself. Further tests have to be performed with larger knowledge bases.

6 Conclusions

This work has assessed the applicability of semantic web technologies in security information management systems, providing a way to semantically share information among different security domains. For this, an ontology based on IDMEF has been defined, which can hold all the information of any IDMEF message. To test this ontology, we have also defined and implemented a semantic collaborative SIMS architecture, where each SIMS stores its IDMEF alerts in a knowledge base and can query other SIMS knowledge bases using a SPARQL interface.

The test performed to store alerts showed the times to save such alerts, which can be acceptable for a prototype but not for a production system that receives tens of alerts per second. Thus, some approaches have been done to improve these times. On the one hand, Jena SDB library has been used to optimize the storage of the ontology in a database. On the other hand, the use of SPARQL/Update has been proposed, to limit the saving time to that information contained on each alert. Another improvement has been the parsing of alerts continuously, to avoid launching a Java process each time an IDMEF message arrives the instance generator. In this way, we could reduce the storing time to a half from the initial approach.

With respect to the query modules, we have done preliminary tests with good results. We will generate further tests, modifying the size of the knowledge base to check how the system performs with larger data sets. It is also important to note that the instances of old alerts are periodically deleted from the knowledge base. This avoids its size grow *ad infinitum*.

As another future work, we will study how to do inference with the information contained in the knowledge bases.

Acknowledgements. This work has been done in the framework of the collaboration with Telefónica I+D in the project SEGUR@ (reference CENIT-2007 2004, https://www.cenitsegura.es), funded by the CDTI, Spanish Ministry of Science and Innovation under the program CENIT.

References

1. D. Dubie: Users shoring up net security with SIM. Network World, 30th September 2001.
2. H. Debar, D. Curry, B. Feinstein: The Intrusion Detection Message Exchange Format (IDMEF). IETF Request for Comments 4765, March 2007
3. T. R. Gruber: A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition, Vol. 5, No. 2 (1993) 199-220.
4. J. Undercoffer, A. Joshi, A. Pinkston: Modeling computer attacks: an ontology for intrusion detection. Lecture Notes in Computer Science, Vol. 2820 (2003) pp. 113-135.
5. D. Geneiatakis, C. Lambrinoudakis: An ontology description for SIP security flaws. Computer Communications, Vol. 30, Issue 6 (2007) pp. 1367-1374
6. S. Dritsas, V. Dritsou, B. Tsoumas, P. Constantopoulos, D. Gritzalis: OntoSPIT: SPIT management through ontologies. Computer Communications, Vol. 32, Issue 1 (2009) pp. 203-212.
7. Joseki – A SPARQL Server for Jena, available at http://www.joseki.org/
8. Jena – A Semantic Web Framework for Java , available at http://jena.sourceforge.net/
9. E. Prud'hommeaux, A. Seaborne: SPARQL Query Language for RDF. W3C Recommendation 15 January 2008.
10. SDB - A SPARQL Database for Jena, available at http://jena.sourceforge.net/SDB/
11. D. L. McGuinness, F. van Harmelen: OWL Web Ontology Language Overview. W3C Recommendation 10 February 2004.
12. J. E. López de Vergara, E. Vázquez, A. Martin, S. Dubus, M. N. Lepareux: Use of ontologies for the definition of alerts and policies in a network security platform, Journal of Networks, Vol. 4, Issue 8 (2009) pp. 720-733.
13. J. H. Gennari, M.A. Musen, R.W. Fergerson, W.E. Grosso, M. Crubézy, H. Eriksson, N.F. Noy, S.W. Tu: The evolution of Protégé: an environment for knowledge-based systems development. Int. J. Hum.-Comput. Stud. Vol. 58, Issue 1 (Jan. 2003) pp. 89-123
14. Object Management Group: Ontology Definition Metamodel Version 1.0. OMG document number formal/2009-05-01, May 2009.
15. ARQ - A SPARQL Processor for Jena, available at http://jena.sourceforge.net/ARQ/
16. A. Seaborne, G. Manjunath, C. Bizer, J. Breslin, S. Das, I. Davis, S. Harris, K. Idehen, O. Corby, K. Kjernsmo, B. Nowack: SPARQL Update, A language for updating RDF graphs. W3C Member Submission 15 July 2008.