Semantic Mediation for A Posteriori Log Analysis

Farah Dernaika, Nora Cuppens-Boulahia, Frédéric Cuppens, Olivier Raynaud

Semantic Mediation for A Posteriori Log Analysis

Farah Dernaika
farah.dernaika@imt-atlantique.fr
IMT Atlantique
Cesson-Sévigné, France
Be-Studys
Geneva, Switzerland

Nora Cuppens-Boulahia
IMT Atlantique
Cesson-Sévigné, France

Frédéric Cuppens
IMT Atlantique
Cesson-Sévigné, France

Olivier Raynaud
Université Clermont-Auvergne
LIMOS CNRS UMR 6158, France
Be-Studys
Geneva, Switzerland

ABSTRACT

The a posteriori access control mode consists in monitoring actions performed by users, to detect possible violations of the security policy and to apply sanctions or reparations. In general, logs are among the first data sources that information security specialists consult for forensics when they suspect that something went wrong. One difficult challenge we face when analyzing logs, is the multiple log file formats. However, normalizing logs in one format needs a lot of processing especially because log files usually contain a high volume of data. Our study proposes then to tackle this problem, by leaving the different log formats as they are, and retrieving information from logs by querying them. A semantic mediator makes it possible to inter-operate various sources of information without modifying their internal functioning. It can be responsible for locating data sources, to transmit queries to each source, or from one source to another, to retrieve the queries responses and possibly send them back to other sources. To the best of our knowledge, semantic mediation techniques have been used to share information from heterogeneous data sources, but they were never used in the context of log analysis.

CCS CONCEPTS

• **Behavioral Analysis for Access and Usage Control** → A posteriori log analysis.

KEYWORDS

Logs, Semantic Mediation, Query Rewriting, Access Policy.

ACM Reference Format:
Farah Dernaika, Nora Cuppens-Boulahia, Frédéric Cuppens, and Olivier Raynaud. 2019. Semantic Mediation for A Posteriori Log Analysis. In *Proceedings of the 14th International Conference on Availability, Reliability and Security (ARES 2019) (ARES '19), August 26–29, 2019, Canterbury, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3339252.3340104

ARES '19, August 26–29, 2019, Canterbury, United Kingdom

ACM ISBN 978-1-4503-7164-3/19/08…$15.00
https://doi.org/10.1145/3339252.3340104

1 INTRODUCTION

In certain sensitive environments, such as the health domain, where users are generally trusted and where particular events (such as emergencies) may occur, security controls in the corresponding information systems must not block certain decisions and actions of users. This could have serious consequences. Indeed, it is important to be able to identify and trace these decisions and actions, in order to detect possible breaches of the security policy put in place and set responsibilities. We consider that a security policy of an information system is a set of rules that define access control requirements (permissions, prohibitions) relating to the actions perfomed by a user on this information system.

Intuitively, a priori control of a security policy is a preventive mode in which any attempt to access the information system that violates the security policy is systematically blocked. In contrast, the problematic of the a posteriori control is fairly recent and was introduced by Sandro Etalle and William Winsborogh [28]. The first researches were mainly concerned with the auditing requirements needed to implement the a posteriori control mode [21, 27]. A post-access control mode consists in monitoring actions performed by users, to detect possible violations of the security policy and to apply sanctions or reparations.

Every system generally logs all events associated with its operating system, running applications, and the network to which it is connected. Recognizing the importance of logs, the National Institute of Standards and Technology, USA issued best practices and recommendations for computer security log management [34]. Therefore, logs are among the first data sources that information security specialists consult for forensics when they suspect that something went wrong.

One important issue that makes these investigations doubtful, is that log analysis is based essentially on the expertise of the person who performs it. For example, the system administrator can use a generic list of security checks that is not necessarily adapted to the target system. Thus, several efforts have been made to detect anomalies from logs such as process mining [46], and machine

learning techniques [23] that were also integrated in some log analysis tools e.g Splunk [37]. Yet, these methods always require human intervention for further analysis to decide what is really normal vs. abnormal.
When it comes to access control, the security policy is the judge. Consequently, it is necessary to define a reference format for the security policy, in order to facilitate the detection of potential violations.

Another difficult challenge we face when analyzing logs, is the multiple log file formats. This is generally due to the different log sources such as Application server, Web server, Database, etc. Several log normalization methods have been proposed in the literature like Regular Expressions, Tokenization, Natural Language Processing, and Custom Normalization using log analysis tools e.g logstash [1]. However, having one format of all the logs needs a lot of processing especially because log files usually contain a high volume of data. In consequence, it is a matter to provide simple, efficient, and economical means to access data logs. Ideally, the solution must guarantee different criteria such as system autonomy, scalability, and transparency for accessing data location and format.

Our study proposes then to tackle this problem, by leaving the different log formats as they are, and retrieving information from logs by querying them. A semantic mediator makes it possible to inter-operate various sources of information without modifying their internal functioning. It can be responsible for locating data sources, to transmit queries to each source, or from one source to another, to retrieve the queries responses and possibly send them back to other sources [47]. To the best of our knowledge, semantic mediation techniques have been used to share information from heterogeneous data sources, but they were never used in the context of log analysis.

In this paper, we show how semantic mediation solutions can be used for log analysis, and then be a part of a security mechanism to detect violations of the security policy.

In the following, Section II presents the state of the art, and Section III talks about the preliminaries related to our subject. Next, we expose our approach in Section IV. Section V provides the proof of concept and Section VI opens a discussion. Finally, we conclude in Section VII.

2 STATE OF THE ART

The motivation of using an a posteriori access control model was brought with the difficulty of managing access control in many environments and organizations.

Interestingly, it is a prerequisite to take into account the organization's uses and practices, so that the deployed security solution is not perceived as a constraint for users with a significant risk of rejection. This might be the case for some medical organizations, where several emergency situations may occur.

In an a posteriori access control model, a trust management system is used to ensure that data resources are only provided to users who are subject to penalties in case of violation. This auditing process is conducted using audit proofs such as logs. According to previous a posteriori access control approaches [15, 17], this kind of security control includes three components: *log process*, *log analysis*, and *accountability*.

A number of researches dealt with this type of access control. For instance, [28] provided the APPLE framework, where users are responsible of logging and keeping traces of their actions, and each data item is governed by its own policy label. In [16], the authors proposed an a posteriori auditing framework that includes observability, conclusions, and proof obligations functions, in addition to the implementation of a proof finder and a proof checker. In the healthcare domain, [22] outlined the needed architecture to apply audit-based access control in electronic health record systems, and discussed the advantages and limitations of their proposal. Other efforts in the medical domain were [5] and [7]. These works provided a solution to perform an a posteriori analysis of security rules using ontologies, and adopted the ATNA standard as a log format [6]. However, it is possible to consider other log formats [32]. Moreover, non contextual security rules were modeled, and techniques to extract necessary data from logs for policy violation detection were proposed. Other applications of the a posteriori access control can be also related to business processes [4], and detecting violation of privacy protection rules in social networks [9].

In our approach, we consider that the information system's logs are governed by the security policy that can be represented according to several models such as, Discretionary Access Control (DAC) [45], Mandatory Access Control (MAC)[10], Role Based Access Control (RBAC)[30], Attribute Based Access Control (ABAC)[33], Organisation-based Access Control (OrBAC)[25], and which will automatically detect deviations, and fix responsibilities. Nevertheless, these logs keep traces of all the established events in the information system, and these events differ from one logging source to another. Therefore, each log source may provide a different/same type of information, in a different/same format, and/or in a different/same location. This fact leads to the need of extracting the necessary information from multiple log sources, to analyze them and detect potential violations.

On the other hand, the multiplication of data sources has made it impossible for a monolithic system to assimilate all the information. To overcome this problem, [47] proposed an architectural model, where a software module is responsible for accessing a set of data sources, while providing clients the illusion of using a single information system. This software module is called *mediator*. This mediator becomes semantical when the data represents structured knowledge with formal semantics. As a result, a *semantic mediator* is based on models of knowledge representation that are able to describe, to a certain extent, the semantics conveyed by a piece of information and on tools to compare and unify the information semantics independently of the underlying structures. It is essentially used for *Query Rewriting* [13], where queries are mediated from a single query access point to various data sources. Yet, the notion of semantics of an entity cannot be represented in an absolute way. It only makes sense when an entity is in relation to a particular context that can be represented by a concept map that describes a particular field of application. A concept is generally defined from the content of an ontology, that is a formal description of an abstract and simplified view of the world that one wants to represent.

As for access control, several researches were interested in using semantic mediation solutions, namely for privacy preserving enforcement. For instance, in [11], a Privacy-Preserving Service-Oriented Data Integration System (PAIRSE) was proposed. PAIRSE

only allows access to information to which users are entitled for a given purpose. The queries in this project are resolved by automatically selecting and composing data services, through the use of mature query rewriting techniques to devise a novel service composition algorithm. Furthermore, [19] provided a solution to the problem of allowing interoperation while preserving autonomy and security of the local sources, by using wrappers and a mediator. The authors used query folding to resolve the semantic heterogeneity of the information sources, that was based on manually expressed rules. The work in [40] proposed a Semantic Access Control model (SAC) that extends RBAC by considering the semantics of objects and associates permission with concepts instead of objects. Based on this model, a mediator-based interoperation system (SACE), was introduced to resolve semantic heterogeneity and enable access control in one process. It was also shown that SACE incurs only minor performance degradation in comparison to non-secure interoperation systems. Another effort for enabling privacy-preserving secure semantic access control was PACT [39]. PACT allows sharing of data among heterogeneous databases while providing privacy and confidentiality for metadata. It is a mediator-based solution, incorporating encrypted ontologies, encrypted ontology-mapping tables and conversion functions, encrypted role hierarchies and encrypted queries. The encrypted results of queries are sent directly from the answering system to the requester, bypassing the mediator to further improve the security of the system. One of the distinctive features of PACT is that very few changes to the underlying databases are required. Moreover, [20] showed how the specification and enforcement of authorization can be implemented in federated database systems. The authors in [42] introduced a concept-level semantic access control for the Semantic Web, that deals with how access controlled resources names can be rewritten using other terms subject to logical rules expressed with Web Ontology Language (OWL) [38]. In addition, an ontology-based rights expression language built on top of OWL to express access rights of resources was presented in [43].

However, all these efforts used semantic mediation techniques to enforce the a priori access control. Since we are working on the a posteriori access control, the semantic mediation will be used in a different way, particularly, for extracting information from multiple log sources. This process of information extraction from logs falls under the first component of the a posteriori access control, and which this paper focuses on, that is "log processing". Therefore, we have a different view of "log processing", to be "log query processing", and by "query processing" we mean "query rewriting".

3 PRELIMINARIES

In this section, we will provide a background on the semantic web standards we will rely on to build our semantic mediator.

The Resource Description Framework (RDF) [35] is a W3C "model for data interchange on the Web". RDF represents real world objects and relationships between them, by using URIs. This graph-based representation is often called as a triple, that is the association of a subject, predicate (i.e. property representing the relationship) and an object.

The Web Ontology Language OWL [38] is a family of knowledge representation languages based on Description Logic (DL) [8] with a representation in RDF. It forms an ontology by defining real world concepts, and their relationships in vocabularies. The concepts in an OWL ontology are named as classes, and relationships as properties. Moreover, OWL ontologies include axioms that assert constraints over their concepts and individuals. These axioms can be realized as simple assertions or as simple rules.

In Addition, these ontologies can be queried with SPARQL [41], a standard query language for RDF proposed by the World Wide Web Consortium (W3C).
A SPARQL query consists of triple patterns (RDF triples where each of the subject, predicate and object may be a variable), conjunctions, disjunctions, and optional patterns. The evaluation of the query is based on graph pattern (a set of triple patterns) matching. This graph pattern, located in the "WHERE" clause of the query, is defined recursively and contains triple patterns and SPARQL operators.

4 OUR PROPOSAL

As mentioned earlier, instead of putting all log types in one unique format, we will adopt semantic mediation techniques for query rewriting, to retrieve information from the different log sources. Let $S = \{S_1, S_2, ..., S_n\}$ be the set of log sources, and $f = \{f_1, f_2, ..., f_n\}$ the set of their corresponding formats.

In an a posteriori access control system, policies are checked after granting access to users. Once authenticated, access to information will be governed by an access control policy that is contextual to the application domain. A reconciliation between policy rules and logged actions is then needed, in order to verify whether access rules are fully respected or not. Therefore, queries will be sent automatically from the defined security policy to the logs. Let P be this policy, supposedly represented in an ontological model [31, 44].

A semantic mediator exists between the policy and the logs for query processing. It is used to overcome the semantic heterogeneity of different log sources, by rewriting a request expressed on one source schema into another request expressed on a target schema. This rewriting process is done using previously established semantic correspondences between the different schemas (ontologies in our case). In addition, the process is divided into two stages: semantic query rewriting and syntactic query rewriting.

4.1 Ontologies for a conceptual view of logs

Each log format has its specific fields, which values vary from one event log to another. Considering that these fields are well known, static local ontologies can be created to provide a conceptual view of log sources. These ontologies can be designed by experts to represent the field names managed by each source. Without modifying any S_i, each field name that is a part of f_i, will constitute a concept of a local ontology O_i.
Let $O = \{O_1, O_2, ..., O_n\}$ be the set of local ontologies relating to the log sources in S.
These local ontologies will contain only the main concepts of each log and not the individuals. For example, if we have a database log that contains the following columns: *UserID*, *Action*, and *TimeLogged*, only these concepts will appear in the ontology and not their

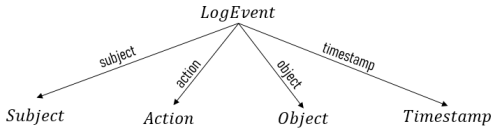

Figure 1: Global Log Ontology O_G.

values e.g *"100"*, *"View"*, and *"2019-02-11 21:31:48"* respectively. We should always remember that the logs will remain intact, and that the ontologies are used for query rewriting purposes.

Nevertheless, one global ontology is needed to represent all log types. As log contents may vary a lot from one source to another, they all have a common thing: all of them simply register the event that occured, more precisely, "what happened? when? by whom?". Therefore, every log type essential element, the *Subject*, *Action*, *Object*, and *Timestamp*, will constitute a concept (a class) of the global ontology. Each class will have a relation with the *LogEvent* class, in which the corresponding attributes appeared. This domain ontology is presented in Figure 1, and is denoted as O_G.

4.2 Mappings between Ontologies

To rewrite a SPARQL query, expressed over O_G, to a SPARQL query expressed over O_i, mappings between O_G and O_i should be established.
A mapping is a set of correspondences between different entities of different ontologies. A correspondence is defined as stated in Definition 1.

Definition 1 (Correspondence) *Let O_1 and O_2 be two ontologies. A correspondence μ is a triplet <e_1, e_2, r> where*

- *e_1 and e_2 are two alignable entities of O_1 and O_2 respectively.*
- *$r \in R$ denotes an existing relation between e_1 and e_2.*

An entity in an ontology can be a class, an object property, a datatype property, or an individual. In our case, individuals don't exist in the ontologies, so there will not be any relative entities. The relationship between entities can be an equivalence (≡) or a subsumption (⊆). Additionally, complex expressions in the correspondences between entities can be found as well, using union (∪) and intersection (∩) operations. For example, μ: O_G:Timestamp ≡ O_1:Date ∪ O_1:Time. As every O_i is static, mappings can be done manually or semi-automatically to set correspondences between each of the concepts *Subject, Action, Object, Timestamp* in O_G, and their relative concepts in O_i.

4.3 Query Rewriting Process

This process is governed by the security policy P, and is about sending SPARQL queries, expressed in terms of the global log ontology O_G, to be transformed subsequently, semantically and syntactically, in the mediator. The resulting queries will be executed on multiple log sources to extract information. We also consider that the initial query Q_G is sent to the log sources that include the requested attribute (e.g concept), to which a mapping was found in the representing ontology.

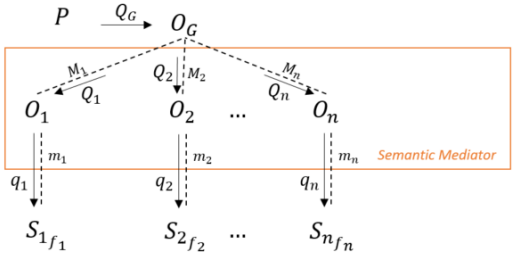

Figure 2: Query Rewriting Process.

4.3.1 Semantic Query Rewriting. The semantic mediator takes Q_G as input, decomposes it into multiple subqueries if needed, and rewrites it (or its subqueries) to a semantically corresponding SPARQL query Q_i, for each O_i, with respect to the mapping M_i that exists between O_G and O_i. We define *SP* as the domain of SPARQL queries, M as the domain of mappings between O_G and O_i, and *SemRW* as the function responsible of the semantic rewriting of a SPARQL query.
SemRW: $SP \times M \rightarrow SP$, $(Q_G, M_i) \rightarrow SemRW(Q_G, M_i) = Q_i$
The rewritten query is generated by replacing the graph pattern of the initial query with the rewritten graph pattern. Variables appearing in the rewritten graph pattern are the same as the variables that appeared in the initial graph pattern. In addition, the rewriting process is independent of the query type (i.e., Select, Ask, etc.), the SPARQL solution sequence modifiers (i.e., Order By, Distinct, etc.) and the SPARQL algebra operators (i.e., Union, Optional, etc.). Since a lot of works treated the SPARQL rewriting problem, we refer to [36] for more rewriting rules details.

4.3.2 Syntactic Query Rewriting. In this second step, a syntactic transformation of the rewritten SPARQL queries (each Q_i) will be achieved.
Different concepts can be used to structure the information in log files such as relationship in the relational model, XML tag, CSV, etc. Thus, the SPARQL query can be converted to an SQL query, XQuery, or any other type of query depending on the existing log formats. Let *QR* be the domain of all query types excluding SPARQL. We define *SynRW* the function for syntactically rewriting a SPARQL query as follows.
SynRW: $SP \times f \rightarrow QR$, $(Q_i, f_i) \rightarrow SynRW(Q_i, f_i) = q_i$ knowing that q_i is understandable by f_i
For each log storage format, specific algorithms for syntactically rewriting SPARQL should be defined. Moreover, mappings m_i between log sources and their corresponding local ontologies can also exist depending on the rewriting algorithm, and the source's type format. These mappings can be manually specified.
Finally, q_i will be executed on S_i, and all the obtained answers will be combined to respond to the initial Q_G.

The proposed solution is presented in Figure 2. Without loss of generality, we will treat the case of two log formats in the rest of the paper, logs in the relational model and in XML, as the corresponding syntactic rewriting algorithms of SPARQL already exist in the literature [12, 26].

4.4 Policy Reconciliation

From the obtained query results, corresponding axioms and assertions will be generated. As any security policy can be represented as a set of quadruples *<subject, action, object, time>*, it is possible to establish links between the query responses and the security attributes used to express the access control policy, to check their compliance and detect if there was any violation. In contrast, for an expressive security policy, fetching metadata of the extracted information is also needed [24].

5 PROOF OF CONCEPT

5.1 Scenarios

5.1.1 Scenario 1. Two hospitals A and B use an Electronic Health Record (EHR) application to share information between each other. However, the server in hospital B generates logs in a database table, while hospital A's server generates XML logs. The two servers record almost the same information about the users actions in the application domain. Evidently, the users appearing in the logs of each server correspond to the employees of the corresponding hospital. In January 2019, a patient X entered the emergency room in hospital A. In order, to access to his medical record, hospital A asks hospital B to send her the patient's medical history. The patient's designated health care professional (HCP) from hospital B sends the patient's medical record to hospital A. Two weeks later, this same patient went to consult his designated HCP in hospital B, when his HCP noticed that there was something wrong in the prescription given from hospital A.
This fact triggered the investigation process to search for the principal cause of the prescription mistake.

5.1.2 Scenario 2. A certain HCP in hospital B took a 4-day leave from work for illness. In consequence, a substitute HCP was called to replace him during this period. On his return, the HCP would like to know which medical records have been modified during his absence, for patients follow-up reasons.

5.1.3 Scenario 3. Going deeper in scenario 1, the reason why the patient went to consult his HCP in hospital B, was his affection with a very low blood pressure, in addition to a lot of vomiting. The error in the prescription was that the medicine prescribed from hospital A is not compatible with the patient's previously prescribed medicine, when he had a bacterial pneumonia, a less than one month before.

Figures 3 and 4 show excerpts of the logs generated on each hospital's server, supposedly configured by the security administrators of each hospital. Besides, the corresponding generated ontologies and mappings are shown in Figure 5.

```
<transaction id="1378" xmlns:action="http://hospital.com/EHR">
<timelogged>2019-01-07 16:42:30</timelogged>
<loggedInMID>9000000003</loggedInMID>
<Action>ADD</Action>
<transactionCode>2210</transactionCode>
<Resource>APT314450</Resource>
</transaction>

<transaction id="3623" xmlns:action="http://hospital.com/EHR">
<timelogged>2019-01-09 10:03:51</timelogged>
<loggedInMID>9000000003</loggedInMID>
<Action>VIEW</Action>
<transactionCode>900</transactionCode>
<Resource>MR314160</Resource>
</transaction>

<transaction id="3674" xmlns:action="http://hospital.com/EHR">
<timelogged>2019-01-09 10:37:04</timelogged>
<loggedInMID>9000000003</loggedInMID>
<Action>ADD</Action>
<transactionCode>1120</transactionCode>
<Resource>LP314160</Resource>
</transaction>

<transaction id="4229" xmlns:action="http://hospital.com/EHR">
<timelogged>2019-01-10 12:24:38</timelogged>
<loggedInMID>5000000001</loggedInMID>
<Action>VIEW</Action>
<transactionCode>900</transactionCode>
<Resource>MR314160</Resource>
</transaction>
```

Figure 3: XML Log.

TransactionNb	FirstMID	Resource	SecondaryMID	Action	Time
265	8000000011	MR314980	314980	VIEW	2019-01-08 18:32:59
544	9000000013	MR314160	314160	VIEW	2019-01-09 10:15:01
545	9000000013	MR314160	hospA	SEND	2019-01-09 10:15:13
1002	9000000085	MR322660	322660	EDIT	2019-01-10 09:48:27

Figure 4: Database Log.

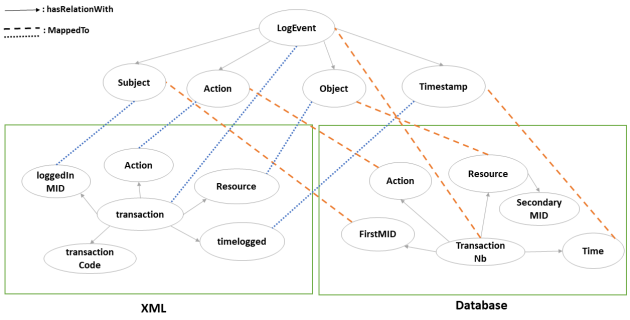

Figure 5: Mappings between ontologies.

5.2 Mediator Implementation

The objective of using a semantic mediator is to enforce the information extraction from logs, in a posteriroi access control. Therefore, we built our semantic mediator by combining different existing open source tools.

To accomplish the semantic rewriting of a SPARQL Query (SPARQL - to - SPARQL), we used a publicly available toolkit for ontological mediation over RDF [2]. This tool rewrites the initial SPARQL query, taking into account the mapping representation, between the global ontology and the different local ontologies, expressed with the Expressive and Declarative Ontology Alignment Language (EDOAL) [18]. EDOAL is a highly expressive and serializable language built upon the Alignment Format [29], a well-known specification extensively used for representing alignments in ontology matching tasks.
However, this toolkit has some limitations since it supports only SELECT and CONSTRUCT queries, and is not able to rewrite the SPARQL query when there is a complex correspondence between

ontologies entities using the union operator. We can overcome this limitation by extending the tool with a function that handles this case. For the sake of simplicity, our defined mappings are currently limited to the exact equivalence of two different entities from two different ontologies.

As for the syntactic query rewriting (SPARQL - to - OtherTypeOfQuery), we were interested in converting SPARQL to SQL and XQuery for test purposes. Many efforts have been made in the literature to do this task, from which we cite [12, 26]. Nevertheless, we relied on open source tools.

For rewriting SPARQL into SQL we used Ontop [14]. Ontop is an open-source Ontology-Based Data Access (OBDA) system that maps data sources to ontologies representing the domain of interest, and through which querying these relational data sources is possible. Advantages of Ontop are its compliance to all relevant W3C recommendations (including SPARQL queries, R2RML mappings, and RDFS ontologies), and its support for all major relational databases. Furthermore, each mapping axiom defined in Ontop corresponds to a pair of source and target. The source is an SQL query over the database, and the target is a graph pattern that contains placeholders that refer to the column names mentioned in the source query. These mapping axioms generate RDF triples, by replacing the placeholders in the target with the values returned when evaluating the source SQL query.

For converting SPARQL to XQuery we used the open-source SPARQLToXQuery [3]. This tool handles only SPARQL SELECT queries in three different cases: (1) the subject and object are variables, (2) the subject is a variable and the object is a literal, and (3) the subject is a variable and the object is an URI. The fact that it only allows the subject of a triple pattern to be a variable, makes the Object and Datatype properties correspond to a subchild of an element in the XML file. Thus, the domain of the property will refer to the parent element, and the range will correspond to its subchild value. It is also worth to mention that the SPARQLToXQuery tool is made to address RDF/XML data. We modified it so that it queries XML.

Figure 6 shows our open-source based semantic mediator architecture.

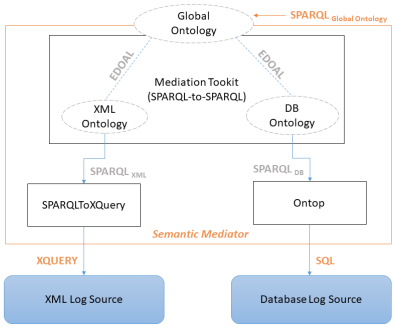

Figure 6: Semantic Mediator Architecture.

5.3 Query Rewriting Applied in the Scenarios

Starting with *Scenario 1*, and considering that the patient's medical id (MID) is 314160, the investigation consists of searching for the actions done, by which subjects, in January 2019, on this patient's medical record. The medical record is noted as "MR314160".

A query rewriting example for this investigation is shown in Table 1 and is explained below.

The initial SPARQL query is transformed into a conjunction of SPARQL queries expressed in terms of the local ontologies. For instance, the object properties *action*, *subject*, *timestamp*, *object* from the global ontology are mapped to the object properties *Action*, *loggedInMID*, *timelogged*, *Resource* and *action*, *executedBy*, *executedAt*, *executedOn* from the ontologies representing the XML log and the Database log respectively. Afterwards, each of the resulted SPARQL queries will be syntactically transformed depending on the underlying log structure. From SPARQL to XQuery, the *transaction* class of the XML ontology refers to the *transaction* element of the XML log and the object property *loggedInMID* refers to the subchild *loggedInMID* of the element *transaction*.

Besides, the other SPARQL query is converted to an SQL query, based on the mappings defined in Ontop. Excerpts of these mappings are shown in Figure 7.

```
mappingId   Resource
target      :{Resource} a :Resource  .
source      select "Resource" from "table_log"

mappingId   M1
target      :{TransactionNb} :executedAt :{Time}.
source      select "TransactionNb", "Time" from "table_log"


mappingId   M2
target      :{TransactionNb} :action :{Action}.
source      select "TransactionNb", "Action" from "table_log"
```

Figure 7: Mappings defined in Ontop.

As for the second scenario, we suppose that the medical ID (MID) of the substituting HCP is "9000000085". Thus, the query is about retrieving the resources that this HCP has edited. Using the same mappings as scenario 1, the SPARQL query is subsequently rewritten semantically and syntactically. Since both HCPs executed their actions in Hospital B, it is obvious to not get an answer from the source log of Hospital A. The query rewriting process of this scenario is shown in Table 2.

We note that log, db, xml shown in the tables refer to the prefix URI of each ontology.

Moving on to the third scenario, we consider that the logs have a finer granularity where the medicines prescribed are logged too, and that more complex mappings are defined between the ontologies (e.g the class Object in O_G is mapped to more than one class in O_i). The query consists then of searching for the doctors who prescribed the conflicting medicines, medicine 1 (med1) and medicine 2 (med2), for this patient, on a 2 month period. We also consider that a query decomposition layer is added to the mediator, which will be used before performing any rewriting. Therefore, the corresponding query of this investigation will be:

```
SELECT ?x ?y ?z WHERE {
?t log:action ?x.
```

```
?t log:object log:MR314160.
{?t log:object log:med1.}
UNION
{?t log:object log:med2.}
?t log:timestamp ?z.
FILTER regex(?z, "^(2018-12|2019-01)")}
```

And will be decomposed into two queries, each one relating to one medicine: $SPARQL_1 \cup SPARQL_2$ where $SPARQL_k$=

```
SELECT ?x ?y ?z WHERE {
?t log:action ?x.
?t log:object log:MR314160.
?t log:object log:med_k.
?t log:timestamp ?z.
FILTER regex(?z, "^(2018-12|2019-01)")}
```

These resulting subqueries will be rewritten according to the different defined mappings and will be sent to each log source, that has the concept medicine. Due to space limitation, the whole query rewriting process of this scenario is not shown, but it will be similar to the one shown in Table 1.

The obtained answers can form quadruples <*subject, action, object, time*>, to compare them with the rules defined in the security policy, and detect possible violations. However, if the security policy is modeled with a higher level of expressivity, for example, according to ABAC or OrBAC, we will need to enrich these results with more attributes. For instance, in scenario 1, the LDAP directory can be consulted to check the roles associated with the extracted MIDs. Therefore, a possible violation can be that the medical record of the patient was consulted and edited by a Lab Technician, who is not supposed to be allowed to do that. As for scenario 2, we can fetch in a database to see if the modified medical records are not related to other than the patients who had an appointment during that period of time.
Finally, decisions can be taken to apply sanctions and reparations or not.

6 DISCUSSION

Every a posteriori access control is built on the base of log processing, more precisely, extracting information from logged data. It is a very important step, since it is the starting point from which the analysis begins, to lead to decisions and set responsibilities. Thus, the use of semantic mediation techniques to accomplish this mission offers many advantages that we detail below.

To start with, it is *economical in terms of processing*. Unlike the existing log management tools, our approach neither parses nor filters provenance logs. The only process it has is the Query Rewriting process, which is quite fast since only one query is handled at a time. The duration of query rewriting and execution is in the range of 300 ms, which is evidently less than any parsing time that varies relatively to the log file size.

Next, it provides *scalability*. Our model is scalable since each data source is autonomous and independent from the other sources. New data sources can be added to the model. As the use case showed how the approach can work for both XML and Database logs, other log formats could be considered. For instance, for a CSV file, we will need to implement a SPARQL to R rewriting algorithm to fulfill the need. However, this current architecture can support CSV files since they can be queried with SQL using specific (java) libraries. One limitation can be that this approach is only suitable for structured or semi-structured log files, since ontologies and mappings have to be defined in advance.

Moreover, the use of SPARQL as a query language enables us to reap the benefits of *federation*, thereby it makes all the log sources look like one big database. Representing the different log formats in RDF serves as a standard *lingua franca* (least common denominator). As such, querying RDF with SPARQL hides the details of a source's particular data structure. This *reduces costs* and *increases robustness* of our model that issues queries. Furthermore, SPARQL enables specific questions to be sent to the logs to *retrieve* directly the *precised information* instead of sending queries with limited number of operations to get an answer.

Besides, the use of the semantic mediation solves the problem of the disparity of the multiple log sources, and makes them interoperable.

Last but not least, our proposal satisfy the requirements of the environment in which the a posteriori access control is deployed, such as the *end-to-end policy enforcement*. It is an end-to-end like question/answer system, from the security policy to the logs. All the query treatments are done transparently in the semantic mediator.

7 CONCLUSIONS AND FUTURE WORKS

In this paper, we proposed a new solution for an a posteriori log analysis based on a semantic mediator. We pictured how it can enforce information extraction from multiple log sources. Besides, we built our approach from existing open source tools. Despite the limitations that they imposed, we showed how our idea can be efficient and economical by testing it on both Database and XML logs.

In connection with this study, our future works consist of enriching the obtained results by fetching information from other types of data sources (other than logs), hence, it will be possible to detect violations of an expressive security policy (e.g ABAC or OrBAC). In addition to the violation detection mechanism.

8 ACKNOWLEDGMENTS

This research is funded by *be-studys*, Meyrin 123, c/o BDO SA, 1219 Châtelaine, GENEVE, a mark of the group *be-ys* dedicated to research and innovation.

REFERENCES

[1] [n. d.]. Elasticsearch Logstash. https://www.elastic.co/products/logstash.
[2] [n. d.]. Mediation toolkit. https://github.com/correndo/mediation.
[3] [n. d.]. SparqlToXQuery. https://sourceforge.net/projects/sparqltoxquery/.
[4] Mohamed Karim Aroua and Belhassen Zouari. 2012. Modeling of A-Posteriori Access Control in Business Processes. In *2012 IEEE 36th Annual Computer Software and Applications Conference Workshops*. IEEE, 403–408.
[5] Hanieh Azkia, Nora Cuppens-Boulahia, Frédéric Cuppens, and Gouenou Coatrieux. 2010. Reconciling IHE-ATNA profile with a posteriori contextual access and usage control policy in healthcare environment. In *2010 Sixth International Conference on Information Assurance and Security*. IEEE, 197–203.
[6] Hanieh Azkia, Nora Cuppens-Boulahia, Frédéric Cuppens, and Gouenou Coatrieux. 2011. A posteriori access and usage control policy in healthcare environment. *Journal of information assurance and security (JIAS)* 6, 192 (2011), 389–397.
[7] Hanieh Azkia, Nora Cuppens-Boulahia, Frédéric Cuppens, and Gouenou Coatrieux. 2012. Ontology based log content extraction engine for a posteriori security control. *Studies in health technology and informatics* 180 (2012), 746–750.

Table 1: SPARQL Rewriting Process in Scenario 1

<table>
<tr><th colspan="2">Original SPARQL Query</th></tr>
<tr><td colspan="2">SELECT ?x ?y ?z WHERE
{?t log:action ?x;
log:subject ?y;
log:timestamp ?z.
Filter regex(?z, "^2019-01")
?t log:object log:MR314160. }</td></tr>
<tr><th>Rewritten SPARQL with XML Mappings</th><th>Rewritten SPARQL with DB Mappings</th></tr>
<tr><td>SELECT ?x ?y ?z WHERE
{ ?t xml:Action ?x ;
xml:loggedInMID ?y ;
xml:timelogged ?z ;
xml:Resource xml:MR314160 .
FILTER regex(?z, "^2019-01")|
}</td><td>SELECT ?x ?y ?z WHERE
{ ?t db:action ?x;
db:executedBy ?y ;
db:executedAt ?z;
db:executedOn db:MR314160 .
FILTER regex(?z, "^2019-01")
}</td></tr>
<tr><th>Generated XQuery</th><th>Generated SQL Query</th></tr>
<tr><td>import module namespace rdffunc;
let $ts := doc('log.xml')//*
for $t in $ts
let $xs:=$t/Action
for $x in $xs
let $ys:=$t/loggedInMID
for $y in $ys
let $zs:=$t/timelogged
for $z in $zs
where $t/Resource='MR314160' and matches($z,"^2019-01")
return <result>
{rdffunc:objectResult($x,$xs)}
{rdffunc:objectResult($y,$ys)}
{rdffunc:objectResult($z,$zs)}
</result></td><td>Select Action, FirstMID, Time
FROM table_log
WHERE Time REGEXP '^2019-01'
AND Resource= 'MR314160';</td></tr>
<tr><th>Query Response</th><th>Query Response</th></tr>
<tr><td><result>
<literal>VIEW</literal>
<literal>9000000003</literal>
<literal>2019-01-09 10:03:51</literal>
</result>
<result>
<literal>VIEW</literal>
<literal>5000000001</literal>
<literal>2019-01-10 12:24:38</literal>
</result></td><td>db:VIEW,
db:9000000013,
db:2019-01-09 10:15:01,

db:SEND,
db:9000000013,
db:2019-01-09 10:15:13,</td></tr>
</table>

[8] Franz Baader, Diego Calvanese, Deborah McGuinness, Peter Patel-Schneider, and Daniele Nardi. 2003. *The description logic handbook: Theory, implementation and applications.* Cambridge university press.

[9] Leila Bahri, Barbara Carminati, and Elena Ferrari. 2015. CARDS-collaborative audit and report data sharing for a-posteriori access control in DOSNs. In *2015 IEEE Conference on Collaboration and Internet Computing (CIC).* IEEE, 36–45.

[10] D Bell, Leonard J LaPadula, M Ben-Ari, G Benson, G Benson, B Appelbe, I Akyildiz, C Date, D Denning, P Denning, et al. 1988. Secure computer system unified exposition and multics interpretation. *Commun. ACM* 1 (1988), 271–280.

[11] Djamal Benslimane, Mahmoud Barhamgi, Frédéric Cuppens, Franck Morvan, Bruno Defude, Ebrahim Nageba, Michael Mrissa, Francois Paulus, Stephane Morucci, Nora Cuppens, et al. 2013. PAIRSE: a privacy-preserving service-oriented data integration system. *ACM SIGMOD Record* 42, 3 (2013), 42–47.

[12] Nikos Bikakis, Chrisa Tsinaraki, Ioannis Stavrakantonakis, Nektarios Gioldasis, and Stavros Christodoulakis. 2015. The SPARQL2XQuery interoperability framework. *World Wide Web* 18, 2 (2015), 403–490.

[13] Béatrice Bouchou and Cheikh Niang. 2014. Semantic mediator querying. In *Proceedings of the 18th International Database Engineering & Applications Symposium.* ACM, 29–38.

[14] Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. 2017. Ontop: Answering SPARQL queries over relational databases. *Semantic Web* 8, 3 (2017), 471–487.

[15] JG Cederquist, R Conn, MAC Dekker, Sandro Etalle, and JI Den Hartog. 2005. An audit logic for accountability. In *Sixth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'05).* IEEE, 34–43.

[16] JG Cederquist, RJ Corin, MAC Dekker, Sandro Etalle, Jeremy den Hartog, and Gabriele Lenzini. 2006. The audit logic: Policy compliance in distributed systems. (2006).

Table 2: SPARQL Rewriting Process in Scenario 2

Original SPARQL Query	
SELECT ?x WHERE {?t log:action log:EDIT; log:subject log:9000000085; log:object ?x. }	
Rewritten SPARQL with XML Mappings	**Rewritten SPARQL with DB Mappings**
SELECT ?x WHERE { ?t xml:Action xml:EDIT ; xml:loggedInMID xml:9000000085; xml:Resource ?x . }	SELECT ?x WHERE { ?t db:action db:EDIT; db:executedBy db:9000000085; db:executedOn ?x . }
Generated XQuery	**Generated SQL Query**
import module namespace rdffunc; let $ts := doc('log.xml')//* for $t in $ts let $xs:=$t/Resource for $x in $xs where $t/Action='EDIT' and $t/loggedInMID='9000000085' return <result> {rdffunc:objectResult($x,$xs)} </result>	Select Resource FROM table_log WHERE Action='EDIT' AND FirstMID='9000000085';
Query Response	**Query Response**
NO ANSWER	db:MR322660,

[17] Ricardo Corin, Sandro Etalle, Jeremy den Hartog, Gabriele Lenzini, and I Staicu. 2004. A logic for auditing accountability in decentralized systems. In *IFIP World Computer Congress, TC 1*. Springer, 187–201.

[18] Jérôme David, Jérôme Euzenat, François Scharffe, and Cássia Trojahn dos Santos. 2011. The alignment API 4.0. *Semantic web* 2, 1 (2011), 3–10.

[19] Steven Dawson, Shelly Qian, and Pierangela Samarati. 2000. Providing security and interoperation of heterogeneous systems. In *Security of Data and Transaction Processing*. Springer, 119–145.

[20] Sabrina De Capitani di Vimercati and Pierangela Samarati. 1997. Authorization specification and enforcement in federated database systems. *Journal of Computer Security* 5, 2 (1997), 155–188.

[21] M A C Dekker and S Etalle. 2007. Audit-Based Access Control for. *Electronic Notes in Theoretical Computer Science* 168, 1 (2007), 221–236. https://doi.org/10.1016/j.entcs.2006.08.028

[22] Mari Antonius Cornelis Dekker and Sandro Etalle. 2007. Audit-based access control for electronic health records. *Electronic Notes in Theoretical Computer Science* 168 (2007), 221–236.

[23] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. DeepLog : Anomaly Detection and Diagnosis from System Logs through Deep Learning. (2017), 1285–1298.

[24] Gilles Dubois and Danielle Boulanger. 2000. A Multi-agent system using semantic metadata for the cooperation among multiple information sources. In *4th European Conf. on Principles and Practice of Knowledge Discovery in Databases*.

[25] Anas Abou El Kalam, Rania El Baida, Philippe Balbiani, Salem Benferhat, Frédéric Cuppens, Yves Deswarte, Alexandre Miege, Claire Saurel, and Gilles Trouessin. 2003. Or-BAC: un modèle de contrôle d'accès basé sur les organisations. *Cahiers francophones de la recherche en sécurité de l'information* 1 (2003), 30–43.

[26] Brendan Elliott, En Cheng, Chimezie Thomas-Ogbuji, and Z Meral Ozsoyoglu. 2009. A complete translation from SPARQL into efficient SQL. In *Proceedings of the 2009 International Database Engineering & Applications Symposium*. ACM, 31–42.

[27] Sandro Etalle, Fabio Massacci, and Artsiom Yautsiukhin. [n. d.]. The Meaning of Logs. ([n. d.]).

[28] Sandro Etalle and William H Winsborough. [n. d.]. A Posteriori Compliance Control Categories and Subject Descriptors. ([n. d.]), 11–20.

[29] Jérôme Euzenat. 2004. An API for ontology alignment. In *International Semantic Web Conference*. Springer, 698–712.

[30] David Ferraiolo, Janet Cugini, and D Richard Kuhn. 1995. Role-based access control (RBAC): Features and motivations. In *Proceedings of 11th annual computer security application conference*. 241–48.

[31] Tim Finin, Anupam Joshi, Lalana Kagal, Jianwei Niu, Ravi Sandhu, William Winsborough, and Bhavani Thuraisingham. 2008. R OWL BAC: representing role based access control in OWL. In *Proceedings of the 13th ACM symposium on Access control models and technologies*. ACM, 73–82.

[32] Bill Gregg, Horacio D'Agostino, and Eduardo Gonzalez Toledo. 2006. Creating an IHE ATNA-based audit repository. *Journal of digital imaging* 19, 4 (2006), 307–315.

[33] Vincent C Hu, David Ferraiolo, Rick Kuhn, Arthur R Friedman, Alan J Lang, Margaret M Cogdell, Adam Schnitzer, Kenneth Sandlin, Robert Miller, Karen Scarfone, et al. 2013. Guide to attribute based access control (abac) definition and considerations (draft). *NIST special publication* 800, 162 (2013).

[34] Karen Kent and Murugiah Souppaya. 2006. Guide to computer security log management. *NIST special publication* 92 (2006).

[35] Graham Klyne and Jeremy J Carroll. 2006. Resource description framework (RDF): Concepts and abstract syntax. (2006).

[36] Konstantinos Makris, Nektarios Gioldasis, Nikos Bikakis, and Stavros Christodoulakis. 2010. Sparql rewriting for query mediation over mapped ontologies. *Technical University of Crete* (2010).

[37] Michael Mayhew, Michael Atighetchi, Aaron Adler, and Rachel Greenstadt. [n. d.]. Use of Machine Learning in Big Data Analytics for Insider Threat Detection. ([n. d.]).

[38] Deborah L McGuinness, Frank Van Harmelen, et al. 2004. OWL web ontology language overview. *W3C recommendation* 10, 10 (2004), 2004.

[39] Prasenjit Mitra, Chi-Chun Pan, Peng Liu, and Vijayalakshmi Atluri. 2006. Privacy-preserving semantic interoperation and access control of heterogeneous databases. In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*. ACM, 66–77.

[40] Chi-Chun Pan, Prasenjit Mitra, and Peng Liu. 2006. Semantic access control for information interoperation. In *Proceedings of the eleventh ACM symposium on Access control models and technologies*. ACM, 237–246.

[41] Eric Prud, Andy Seaborne, et al. 2006. Sparql query language for rdf. (2006).

[42] Li Qin and Vijayalakshmi Atluri. 2003. Concept-level access control for the semantic web. In *Proceedings of the 2003 ACM workshop on XML security*. ACM, 94–103.

[43] Yuzhong Qu, Xiang Zhang, and Huiying Li. 2004. OREL: an ontology-based rights expression language. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. ACM, 324–325.

[44] Nitin Kumar Sharma and Anupam Joshi. 2016. Representing attribute based access control policies in owl. In *2016 IEEE Tenth International Conference on Semantic Computing (ICSC)*. IEEE, 333–336.

[45] Sabrina De Capitani di Vimercati. 2011. *Discretionary Access Control Policies (DAC)*. Springer US, Boston, MA, 356–358. https://doi.org/10.1007/978-1-4419-5906-5_817

[46] Jacques Wainer. [n. d.]. Anomaly Detection using Process Mining. ([n. d.]), 1–13.

[47] Gio Wiederhold. 1992. Mediators in the architecture of future information systems. *Computer* 25, 3 (1992), 38–49.