Alohaly *et al. Cybersecurity* (2019) 2:2
https://doi.org/10.1186/s42400-018-0019-2

Cybersecurity

RESEARCH Open Access

Automated extraction of attributes from natural language attribute-based access control (ABAC) Policies

Manar Alohaly[1,2*], Hassan Takabi[1] and Eduardo Blanco[1]

Abstract

The National Institute of Standards and Technology (NIST) has identified natural language policies as the preferred expression of policy and implicitly called for an automated translation of ABAC natural language access control policy (NLACP) to a machine-readable form. To study the automation process, we consider the hierarchical ABAC model as our reference model since it better reflects the requirements of real-world organizations. Therefore, this paper focuses on the questions of: how can we automatically infer the hierarchical structure of an ABAC model given NLACPs; and, how can we extract and define the set of authorization attributes based on the resulting structure. To address these questions, we propose an approach built upon recent advancements in natural language processing and machine learning techniques. For such a solution, the lack of appropriate data often poses a bottleneck. Therefore, we decouple the primary contributions of this work into: (1) developing a practical framework to extract authorization attributes of hierarchical ABAC system from natural language artifacts, and (2) generating a set of realistic synthetic natural language access control policies (NLACPs) to evaluate the proposed framework. Our experimental results are promising as we achieved - in average - an F1-score of 0.96 when extracting attributes values of subjects, and 0.91 when extracting the values of objects' attributes from natural language access control policies.

Keywords: Attribute-based access control (ABAC) policy authoring natural language processing relation extraction clustering deep learning

Introduction

The concept of access control mechanisms has been around since Lampson's access matrix was coined in the late 1960s (Lampson 1974). Thereafter, dozens of models have been proposed upon the shortage of others. Most of these models, however, have shown forms of inadequacy in withstanding the ever-increasing complexity of today's business environments or safeguard the compliance demand of dynamic organizations. From within this dilemma, attribute-based access control (ABAC) has emerged as a good fit.

The National Institute of Standards and Technology (NIST) Special Publication 800-162 "Guide to attribute-based access control (ABAC) Definition and Considerations" testifies to ABAC's potential to promote information sharing among enterprises with a heterogeneous business nature (Hu et al. 2013). In fact, it is predicted that "by 2020, 70% of enterprises will use attribute-based access control... as the dominant mechanism to protect critical assets" (Gartner 2013). Despite the promising potential, the ABAC policy authoring task has stood out, among other factors, as a costly development effort. This is anything but a new challenge. In fact, the complex syntax of what has become the standard ABAC policy language, namely XACML (OASIS 2013), has been well understood since the earliest days of ABAC. Hence, it is known since 2003 that "XACML is intended primarily to be generated by tools" (McCarthy 2003). This is where the policy authoring tools come into play.

XACML is a generic policy language framework ratified by OASIS to provide an abstraction layer for defining access control policies (ACPs) as machine-readable

*Correspondence: Manaralohaly@my.unt.edu
[1]Department of Computer Science and Engineering, University of North Texas, Denton, TX, USA
[2]College of Computer and Information Sciences, Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia

authorization rules (OASIS 2013). It promotes standardization, expressiveness, granularity, interoperability, and efficiency (Abassi et al. 2010). This flexibility, on the other hand, has introduced implementation complexity. Solutions proposed to aid in the ABAC policy authoring task have ranged from GUI policy authoring tools to APIs and plug-ins (Turner 2017; Axiomatics 2017). These tools generally provide a higher abstraction level that obscures much of XACML's complex syntax. The former set of tools, i.e., the GUI aids, are designed as fill-in forms or spreadsheets that present policy authors with lists, or other graphical formats, populated with valid options of policy elements (i.e., subjects and objects attributes). The assumption is that a non-IT specialist would easily be able to construct a policy by selecting its constituent attributes from drop-down menus (Turner 2017). The latter set of tools is mainly designed for developers as it borrows much of its structure from common programming languages (Axiomatics 2017). Either way, the goal is to automatically generate XACML equivalent policies with the help of the tools. While such aids can provide a better authoring experience, they heavily rely on back-end attribute exchange providers. This strong assumption raises the question of: *where do the required attributes come from?* Upon closer inspection of the ABAC authorization policy lifecycle (Fig. 1), policies are introduced to a system at the earliest stages of its development lifecycle as use-cases. At this point, access control policies, as well as other security and functional requirements, are components of specifications documents, which are almost always written in natural language. To identify the policy elements required to build the machine-readable authorization rules, the current practice assumes that a security architect or system analyst should manually analyze the existing documentations to extract the relevant information (Brossard et al. 2017). The fact that these requirements are often elicited from a pool of natural language statements, e.g., interview transcripts, blurs the boundary between the policies and other requirements. Hence, manually deriving the attributes along with other policy elements needed to feed the previously mentioned policy authoring tools is rather a repetitive, time-consuming and error-prone task.

The necessity to automate this manual process has motivated several studies in developing automated tools to derive elements of access control rules directly from natural language policies (Xiao et al. 2012; Slankas and Williams 2013; Slankas et al. 2014; Narouei et al. 2017; Narouei and Takabi 2015a). Prior research efforts have mostly been restricted to two areas: (1) automatic identification of the access control policy sentences from natural language artifacts, and (2) automatic extraction of the subject, object and action triples from these sentences. In our previous work (Alohaly et al. 2018), we took a step to address the automation task in the context of a flat ABAC system, where the concept of hierarchy

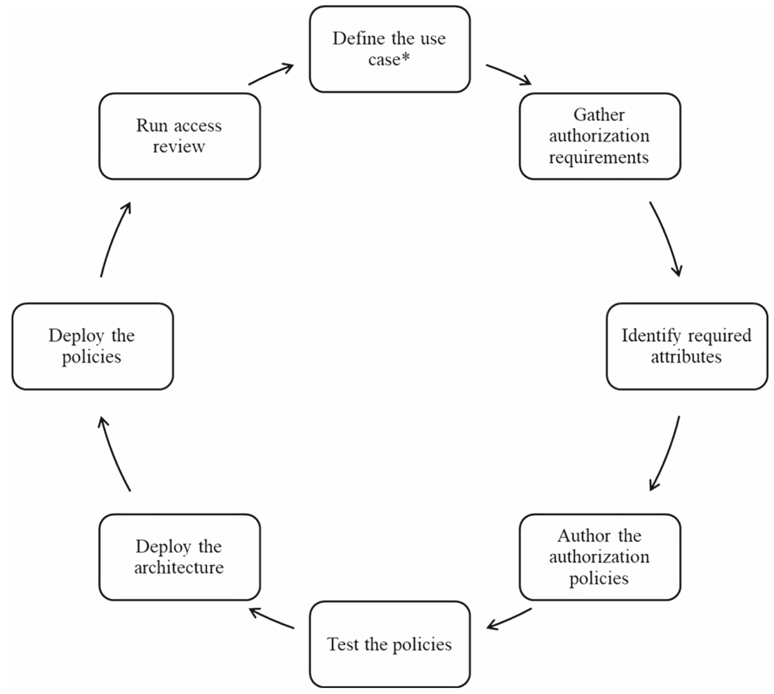

Fig. 1 The authorization policy lifecycle (adopted from Brossard et al. (2017)). *Asterisk marks the beginning of the cycle

among subject or object elements was not considered. Particularly, we designed a framework, consisting of four modules, to extract authorization attributes from NLACPs. However, we mainly focused on one module of the proposed framework, namely attribute extraction, which detects values of attributes in sentences defining policies. In this work, we present the functionalities of other modules required to map attribute values to attribute keys, i.e., to infer the attributes key/value pairs, which entails identifying values that constitute a value space of each attribute. In addition, in this paper the extraction scenario is framed around the hierarchical ABAC model, as opposed to the flat model, to better meet the requirements of real-world organizations. Therefore, we extend the original framework to capture the hierarchical structure of ABAC systems given the NLACPs.

The remainder of this paper is organized as follows: "Background" section provides background information. In "The proposed methodology" section, we present the proposed framework and discuss our methodology to automate the attributes extraction task. "Corpus creation" section describes the procedure to construct the required corpus. We report our experimental results and discuss the limitations of the proposed approach in "Experimental results and performance evaluation" section. We review the related work in "Related work" section. In "Conclusions and future work" section, we conclude our study with the recommendations for future work.

Background

In the following subsections, we define key terminology used in this study. Then, we provide background information regarding ABAC policy authoring process and the underlying techniques of our proposed framework.

ABAC definition and terminology

In its basic form, ABAC is defined as "an access control method where subject requests to perform operations on objects are granted or denied based on assigned attributes of the subject, assigned attributes of the object..." (Hu et al. 2013). We refer to the basic form of ABAC as a flat ABAC model since it does not consider hierarchies. Daniel et al. have introduced the concept of *hierarchical groups* to this basic definition of ABAC to enable attributes inheritance through subject and object groups (Servos and Osborn 2015). The following list provides a high-level definition of terms relevant to our study. We also exemplify each element using the illustrative example shown in Fig. 2, whenever possible. For detailed explanation of the basic and the hierarchical forms of ABAC, we refer the reader to Hu et al. (2013); Servos and Osborn (2015).

- **Subject**: is an entity that initiates access requests to perform actions on objects, e.g. a user whose "subject_type" is "nurse" (Hu et al. 2013).
- **Subject group**: is a set of subjects grouped according to their domain (Servos and Osborn 2015). Using our example in Fig. 2, "subject" represents a group of all subjects or users in the system while "employee" is a sub-group of the main group, "subject," as indicated using the fully qualified name. Similarly, "health_professional" is a sub-group of "employee".
- **Subject attribute**: is a characteristic of the subject, e.g. "senior" as a value of the attribute "rank" (Hu et al. 2013). In a hierarchical ABAC system, subject attributes are assigned to individual subjects as well as to subject groups and are allowed to be inherited through the hierarchical group structure (Servos and Osborn 2015). In our example, we have "rank" as an attribute of the groups "employee" and "health_professional" —by inheritance, as the latter is a sub-group of the former— which also has the attribute "working_hours".
- **Object**: is anything upon which an action might be performed by a subject, e.g., an object whose "object_type" is "lab procedure."
- **Object group**: is a set of objects, e.g., the group "object" and its sub-group "lab_procedure" (Servos and Osborn 2015).
- **Object attribute**: is a characteristic of the object, e.g., "approved" as a value of the attribute "status" (Hu et al. 2013). Similar to subject attributes, object attributes are assigned to individual objects and object groups.

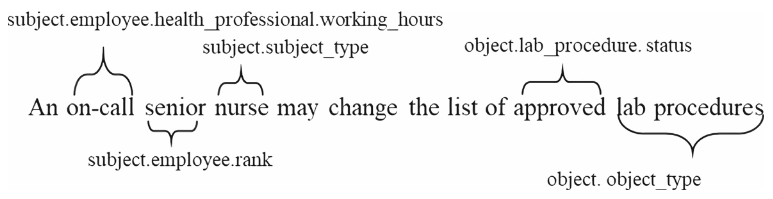
subject.employee.health_professional.working_hours
subject.subject_type
object.lab_procedure. status
An on-call senior nurse may change the list of approved lab procedures
subject.employee.rank
object. object_type

Fig. 2 Attribute identification scenario

Herein, we use the terms "authorization requirement", "access policy" and "access control policy" interchangeably to refer to natural language access control policy (NLACP). We also use "policy elements" as well as "elements" to refer to subject and object elements of the authorization requirement, while we use "attributes" to refer to their corresponding characteristics.

ABAC policy authoring

Brossard et al. have designed a systematic approach for implementing ABAC (Brossard et al. 2017). The authors' proposed model is composed of the iterative sequence of phases shown in Fig. 1. Several studies have researched challenges and opportunities faced by natural language processing (NLP) applications in the automation of the second phase of the cycle, named "gather authorization requirements" (Xiao et al. 2012; Slankas and Williams 2013; Slankas et al. 2014; Narouei et al. 2017; Narouei and Takabi 2015a). Our work complements these prior efforts to aid security architects in the process of deriving required attributes from natural language authorization requirements, which is the third phase of the lifecycle. We, therefore, zoom into the details of this phase to establish an understanding of the problem domain and to identify key activities with the potential to be automated. This, in turn, provides insights into the essential building blocks needed to design a practical ABAC attribute identification framework.

The goal of the "identify required attributes" phase (see Fig. 1) is to identify attributes that jointly build up an ABAC access control rule. Each attribute is defined by the following pieces of information, as suggested by Brossard et al. (2017):

- **Short name**: a key a security architect would use to refer to an attribute while writing policy.
- **Namespace**: the subject or object group to which an attribute is assigned. In this work, we focus on the hierarchical grouping of policy elements. Thus, this property indicates the hierarchical path leading the group of interest.
- **Category**: the class to which an attribute belongs. The core categories are subject, action, and object (OASIS 2013).
- **Data type**: the data type of an attribute. The most common data types are: string, boolean, numerical, and date and time.
- **Value constraints**: the values that can be assigned to an attribute.

Running example

Suppose that a simplified health care system has established the following set of policies:

- An on-call senior nurse may change the list of approved lab procedures.
- A nurse (first-shift or second-shift) may not change the list of approved lab procedures.
- A junior nurse may view the list of pending lab procedures.
- An on-call senior lab technician may request follow-up lab procedures.
- A senior employee may request long-term compensation.
- A registered patient may view his full health record.

Given this set of policies, a security architect should first define authorization attributes based on the previously mentioned properties, as shown in Table 1. Then, information in Table 1[1] is combined to build ABAC rules that are necessary to enforce such authorizations. Figure 3, for instance, presents a machine readable policy equivalent to the first authorization requirement in the list. It is

Table 1 List of attributes needed to build ABAC authorization rules to enforce policies presented in the running example (see "Running example" subsection)

Short name	Namespace	Category	Data type	Values
Subject_type	Subject	Subject_cat	String	Nurse, lab technician, employee, patient
Object_type	Object	Object_cat	String	Lab procedure, compensation, health record
Action_type	Action	Action_cat	String	View, change, request
Rank	Subject.employee	Subject_cat	String	Senior, junior
Working_hours	Subject.employee. health_professional	Subject_cat	String	First-shift, second-shift, on-call
Status	Subject.patient	Subject_cat	String	Registered
Status	Object.lab_procedure	Object_cat	String	Approved, pending, rejected
Type	Object.lab_procedure	Object_cat	String	Follow-up
Status	Object.health record	Object_cat	String	Full
Period	Object.compensation	Object_cat	String	Long-term

The qualified name format, i.e., A.B, used in the namespace column denotes the hierarchical path leading the group that contains the respective attribute

```
namespace subject{
attribute subject_type{
category = subject_cat
id = "subject_type"
type = string}
namespace employee{
attribute rank{
category = subject_cat
id = "rank"
type = string}
namespace health_professional
    {
attribute working_hours{
category = subject_cat
id = "working_hours"
type = string }}}}
namespace object{
attribute object_type{
category = object_cat
id = "object_type"
type = string}
namespace lab_procedure{
attribute status{
category = object_cat
id = status
type = string}}}
```

```
namespace action{
attribute action_type{
category = action_cat
id = "action_type"
type = string}}
policy change{
target clause subject.
    subject_type
=="nurse" and
subject.employee.rank=="senior"
and
subject.employee.
    health_professional.
working_hours == "on-call"
and
object.object_type=="lab
    procedure"
and
object.lab_procedure.status
=="approved" and
action.action_type == "change"
rule allow {permit}}
```

Fig. 3 The machine-readable form of a policy sentence "An on-call senior nurse may change the list of approved lab procedures." Bold-ed portions of text correspond to attribute properties as obtained from Table 1

apparent that by having well-defined attributes, building the machine-readable rule becomes a relatively simple task.

In light of this description, an automated ABAC attribute identification tool infers properties needed to define each attribute. For this purpose, we design a framework, shown in Fig. 4, using natural language processing (NLP), relation extraction (RE) and machine learning (ML) techniques.

Natural language processing techniques

This section provides an overview of the relation extraction, natural language processing and machine learning techniques used in our framework.

Relation extraction (RE)

Relation extraction is a central task in natural language processing. The aim of this task is to extract semantic relation between pairs of co-occurring entities, also known as arguments, mostly within a sentence. Throughout the literature, RE problem has been often addressed as a classification task where the goal is to assign a predefined relation to pairs of arguments (Jiang 2012). This approach is mainly organized in two stages. First is the identification of candidate relation instances, and then the classification of candidate instances into valid and invalid. As a standard classification task, classifiers are built upon training corpus in which all relations and its arguments have been manually annotated. The annotated relations are used as

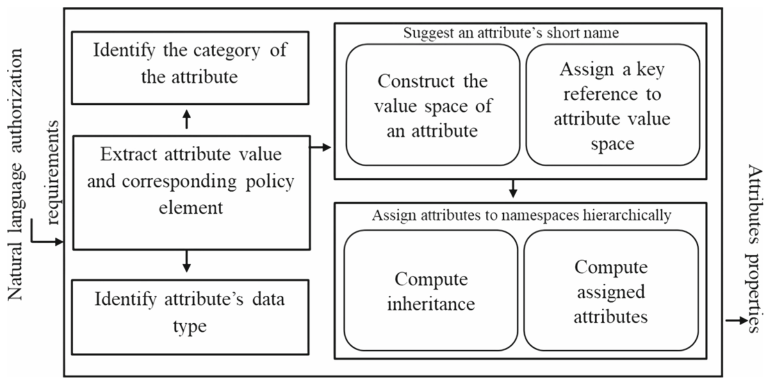
Natural language authorization requirements
Identify the category of the attribute
Extract attribute value and corresponding policy element
Identify attribute's data type
Suggest an attribute's short name
Construct the value space of an attribute
Assign a key reference to attribute value space
Assign attributes to namespaces hierarchically
Compute inheritance
Compute assigned attributes
Attributes properties

Fig. 4 High level overview of the proposed attributes extraction approach

positive training examples, whilst the rest of co-occurring pairs that are not labeled are deemed as negative training examples. Traditional machine learning or deep learning algorithms can then be used to train relation extraction classifiers.

Natural language processing (NLP) in RE

Systems for relation extraction often have -as components-four natural language processing techniques which are: tokenization, Part of Speech (POS) tagging, named entity recognition and syntactic parsing.

Tokenization is an essential starting point for almost any NLP task. It splits text into tokens which are words or punctuation marks (Webster and Kit 1992). Considering English language, rule-based text tokenization, using spaces or punctuations, is a straightforward task. However, abbreviation-like structures introduce a level of complexity to this process.

POS taggers determine the corresponding POS tag for each token in a sentence (Brill 1995). It is a sequential labeling problem. In the context of RE, POS tagger is used to generate candidate instances by matching against predefined POS patterns. POS are also useful in providing lexical information needed to design RE models. In attribute relation extraction, for example, attributes are usually adjectives and policy elements are nouns or combination of nouns.

Named entity recognition (NER) aims to detect phrases that express some real-world entities such as people, organizations, locations, concepts, times and events in a given segment of text (Tjong Kim Sang and De Meulder 2003). Oftentimes, this task requires more than simple matching against predefined dictionaries or gazetteers. Real-world applications, thus, combine heuristics, probabilistic matching, and sequential learning techniques (Martin and Jurafsky 2000). RE has adopted NER to identify arguments of a relation and to encode semantic knowledge of these arguments.

Syntactic parsing techniques provide a middle ground between the lexical representation of a sentence and its meaning. The two dominant syntactic parsing strategies are constituency and dependency parsers (Zelenko et al. 2003; Zhang et al. 2006; Culotta and Sorensen 2004; Jiang 2012). Relation extraction studies have empirically shown that patterns drawn from a dependency parse trees are more resilient to lexical inconsistency among different domains (Johansson and Nugues 2008).

Convolutional neural network (CNN) in RE

CNN has recently been used in solving a wide range of applied machine learning problems. In RE, CNN can be used to model syntactic and semantic relations between words within a sentence while reducing the dependence on manually designed features. Generally speaking, the use of CNN-based models in various NLP tasks has achieved impressive results (Collobert et al. 2011; Kalchbrenner and Blunsom 2013; Zeng et al. 2014).

The architecture of a typical CNN consists of a stack of layers with (1) an input layer that encodes words in each relation instance by real-valued vectors; (2) a convolutional layer to capture contextual features, e.g., n-grams, in the given input; (3) a pooling layer to determine the most relevant features and, (4) an output layer which is a fully connected layer that performs the classification. "Classify candidate instances using CNN-based model" subsection provides a detailed discussion of each layer as well as the overall structure of our CNN.

Word embeddings and compositionality principle

In the last decade, there have been several proposals to generate distributed representations of words often referred to as words embeddings (Mikolov et al. 2013; LeCun et al. 2015). In our work, we do not only need vector representations of individual words, but also of sequence of words, i.e., phrases, to account for multi-word attribute values. Such representations can be obtained using the concept of compositionality, which states that the meaning of an expression is determined as a function of the words it contains (Pelletier 1994). The combined representation of words in a sequence can be obtained using basic algebraic operations as discussed in Mikolov et al. (2013). We have empirically shown that among different algebraic operations, the vector addition works the best for our purpose. This goes in line with the findings discussed in Banea et al. (2014). Therefore, we adopt the additive composition to obtain a single representative meaning of multi-word policy element or attribute values.

DBSCAN clustering algorithm

Density based spatial clustering of application with noise (DBSCAN) is a density-based clustering algorithm. It is a density-based algorithm in the sense that it defines clusters as connected dense regions in the data space (Ester et al. 1996). With this logic, DBSCAN is capable of discovering cliques with arbitrary shape and size without a preset number of clusters. However, it requires two other input parameters, namely *eps* and *MinPts*. Epsilon (*eps*) specifies the neighborhood distance cutoff while *MinPts* defines the minimum number of points needed to create a cluster (Schubert et al. 2017). DBSCAN labels the data points as core points, border points, or outlier points (Tran et al. 2013).

Definition 1 *a core point is any point that has at least MinPts points directly reachable within the neighborhood distance of eps.*

Definition 2 *a border point is a non-core point that has at least one core in its neighborhood.*

Definition 3 *outliers are neither core points nor border points.*

Definition 4 *density-reachability captures the transitive similarity between points. Two points, p_1 and p_n, are called density-reachable if there exists a chain of points p_i, p_{i+1}, ... p_n where $i \geq 1$ and $n \geq 2$ such that for all $i < n$, p_i is a core point and p_{i+1} is a neighbor of p_i while p_n is either a core point or border point. On the basis of these definitions, the goal of DBSCAN is to find some points which satisfy the minimum density requirement of MinPts points within the eps distance to be mareked as cores. Then, expand the cores using transitive similarity to include border points, as illustrated in Algorithm* 1 *(Rehman et al.* 2014*). In our work, constructing the value space of an attribute (see "Constructing the value space of an attribute" subsection) is modeled as an application of DBSCAN.*

Algorithm 1: Pseudo-code of the DBSCAN as discussed in Rehman et al. (2014)

```
Input : @dataset, @eps, @minPts
Output: Clusteres of Dataset
1  Function DBSCAN(dataset, eps, minPts )
2      clusterId = 1
3      for p in dataset do
4          if p is classified then
5              continue
6          if count(epsNeighborhoods(p)) >= minPts
           then
7              setClusterId (p, clusterId)
8              seeds = epsNeighborhoods(p)
9              while seeds is not empty do
10                 point = seeds.pop()
11                 setClusterId (point, clusterId)
12                 if count(epNeighborhoods(point)) >=
                   minPts then
13                     seeds.push(epsNeighborhoods
14                     (point))
15             clusterId +=1
16         if count(epsNeighborhoods(p)) < minPts
           AND core in epsNeighborhoods(p)) then
               // these are the border
                  points
17             clusterId = 0
18             setClusterId (p, clusterId)
19             continue
20         else
               // these are the outlier
                  points
21             clusterId = -1
22             setClusterId (p, clusterId)
23             continue
```

The proposed methodology

As mentioned earlier in the "Background" section, attributes in ABAC model are defined with short name, potential value(s), data type, category and namespace. Out of these five dimensions, the property that is usually expressed in a natural language authorization requirement is the attribute value. Looking back to our illustrative example shown in Fig. 2, *senior*, for instance, refers to the rank of the nurse. Similarly, *approved* describes the status of the lab procedure that can be accessed by the authorized subjects. From the natural language standpoint, phrases that express such characteristics follow certain grammatical patterns to modify particular elements of the policy sentence. Therefore, our approach leverages the grammatical relations between policy elements and constituents that modify these elements, to view this task as a relation extraction problem (GuoDong et al. 2005; Jiang 2012). Extracting the value of an attribute, e.g., *senior*, and associating this value with the corresponding policy element, e.g. *nurse*, equates finding the value and category properties of two attributes, namely rank and subject_type as described later in this section. However, deriving the short name, e.g., rank, the data type, e.g., string, as well as the namespace, e.g., subject.employee, requires further analysis.

Figure 4 shows an overview of our framework where inner rectangles depict the five main modules and arrows point in the direction of data flow. The framework begins with analyzing the natural language authorization requirement to locate modifiers of each policy element. Then, it proceeds towards four other tasks of identifying attributes' category, their data type, suggesting an attribute short name and assigning attributes to namespaces hierarchically.

Module 1: Attribute extraction

We define two relations, named subject-attribute and object-attribute. These relations exist between policy elements and a phrase or clause that expresses a characteristic of this particular element. To capture these relations, we first identify the grammatical patterns that encode each relation. Using the most common patterns, we generate lists of candidate instances, which are then fed into a machine learning classifier to determine whether or not a candidate instance encodes the relation of interest. The following subsections provide the details of each step. It is worth mentioning that capturing relations of interest can be addressed as a multi-class learning task. However, results obtained from this experiment design were less promising. This motivates our design decision of using two separate binary classifiers, one for each relation.

Identify patterns encoding the target relations

An authorization requirement is deemed to involve a subject-attribute relation if it contains at least one modifier that describes a characteristic of an authorized subject. The same applies to object-attribute relation, but with respect to the object element.

Identifying patterns encoding these relations requires a structured representation of the textual authorization requirement. The constituency parse tree and the dependency tree are both valid options. However, relation extraction studies have empirically shown that patterns drawn from the dependency tree are more resilient to lexical inconsistency among different domains (Johansson and Nugues 2008). We, therefore, represent policy sentences with the typed dependency tree as shown in Fig. 5 for the sentence "on-call senior nurse may change the list of approved lab procedures." In this tree, each node represents a word from the sentence, whilst edges represent the grammatical relationship between two words. *Nurse*, for instance, functions as the nominal subject (nsubj) of *change*. The paths that connect *senior* and *on-call* to *nurse* are of special interest as they encode subject-attribute relation. Similarly, the path between *approved* and *procedures* encodes an object-attribute relation. In other words, the shortest dependency path that links subject or object elements with their attributes is deemed as a relation pattern. To extract such patterns, we refer to our manually annotated data in which we explicitly annotate the authorization attributes of both subjects and objects as described in corpus creation, "Corpus creation" section. From the entire set of resulted patterns, we follow the recommendation of Berland et al. to focus on the most frequent ones (Berland and Charniak 1999).

Generate candidate instances

After identifying the most frequent patterns for the target relations, we search through our dataset to find all matching instances. In this study, we target two relations that are encoded with two different sets of patterns. Matching against these patterns produces two lists of candidates instances of subject-attribute and object-attribute relations. For both relations, instances are represented with the tuple R(E,AV). R defines the overall relation. Depending on the pattern used for generating the instance, E, which stands for element, could either be the subject or object. AV is the value of an attribute associated with E. The pair E and AV represent arguments of the relation R. Note that candidate instances do not always encode a valid relation. We, therefore, feed instances to a machine learning classifier(s) to decide whether or not a candidate instance encodes the relation of interest.

Classify candidate instances using CNN-based model

The extracted candidate instances are now used to build the CNN-based classifiers for both relations. Particularly, the concatenation of arguments in each candidate instance along with the heads of arguments is what will be fed to the model, as illustrated in Fig. 6. Since CNNs can only accept fixed length inputs, we compute the maximum length of entity arguments and choose the input width to be more than twice as long as the maximum length (to account for both arguments and the heads). Inputs longer than the predefined length are trimmed, while shorter inputs are padded with special tokens. Figure 6 shows the overall architecture of our CNN. Further, in the followings, we provide the rationale for each layer of the model.

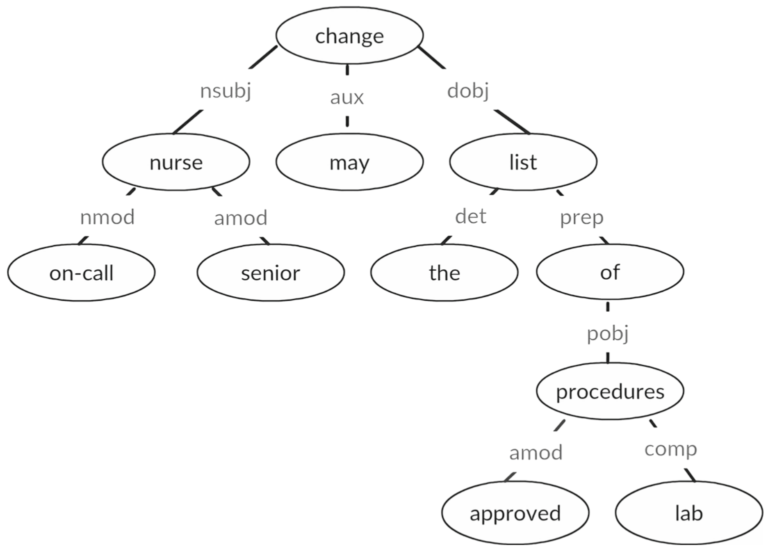

Fig. 5 Dependency tree representation

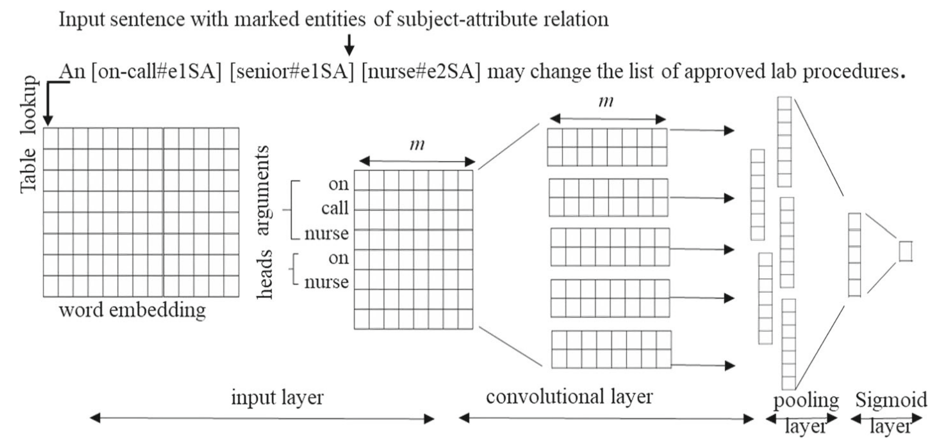
Input sentence with marked entities of subject-attribute relation
An [on-call#e1SA] [senior#e1SA] [nurse#e2SA] may change the list of approved lab procedures.
Table lookup
word embedding
arguments
heads
on
call
nurse
on
nurse
m
m
input layer
convolutional layer
pooling layer
Sigmoid layer

Fig. 6 The architecture of the CNN model. The marked (or annotated) arguments of a subject-attribute instance are obtained as discussed in "Identify patterns encoding the target relations" section. The arguments and their heads are concatenated and fed to the network using word embeddings representation

– **Input layer:** at the very first layer, the two relation arguments as well as their heads are concatenated to form the vector $x = [x_1, x_2, \ldots, x_n]$ where x_i is the i-th word in this sequence, as seen in Fig. 6. Before being fed into the CNN, each word x_i in x must be encoded in a real-valued vector e of m dimensions using word embeddings table **W**. The embeddings table **W** can either be learned as a part of the model or loaded from a pretrained word embeddings. Following the recommendation of Chen et al. (Chen and Manning 2014), which calls for the latter approach, we build our model using the GloVe pretrained word embeddings (Pennington et al. 2017). The output of this layer is a matrix $\mathbf{x} = [x_1, x_2, \ldots, x_n]$. The size of $\mathbf{x}$ is $m \times n$ where m is the dimensionality of the word embeddings vectors, and n is the length of the input sequence x.

– **Convolution layer:** to learn a higher level syntactic and semantic features, the matrix **x**, representing the given input, is passed through a convolution layer with filter $\mathbf{f} = [f_1, f_2, \ldots, f_w]$ of size $m \times w$ where w is the width of the filter. The filter convolves over the input sequence to learn the contextual features from the w adjacent terms in the input. The convolution of the two matrices, **x** and **f**, results in a "windowed" average vector $\mathbf{h} = [h_1, h_2, \ldots, h_{n-w+1}]$ as defined in Eq. 1:

$$h_i = g\left(\sum_{j=0}^{w-1} f_{j+1}^T x_{j+i}^T + b\right) \tag{1}$$

Here g is an optional activation function and b is a bias term. The trained weights of **f** would then be used to detect the features of the relations of interest.

– **Pooling layer:** the pooling layer is mainly used for dimensionality reduction purposes. Its two basic types are the max and average pooling. Max pooling is known to be useful in the relation extraction task as it only passes the salient features of the vector **h** to the subsequent layers, and filters out the less informative ones (Shen et al. 2014). Hence, we apply a max operation to the result of each filter in our model. The application of the max pooling over the vector **h** produces a scalar p = $max(\mathbf{h}) = max\ \{ h_1, h_2, \ldots, h_{n-w+1} \}$

– **Fully connected layer:** in this step, the pooling scores obtained from the previous layer are concatenated into a single feature vector **z**. The vector **z** is then fed into a sigmoid layer to perform the classification task.

Module 2: Suggesting attributes short names

Module 1 is designed to capture the element-value pairs, denoted as (E , AV), given NLACPs. Thus, by running the subject-attribute relation extractor over our running example (see "Running example" subsection) the expected output would be: {(nurse, on-call), (nurse, senior), (nurse, first-shift), (nurse, second-shift), (nurse, junior), (lab technician, on-call), (lab technician, senior), (employee, senior), (patient, registered)}. Each element of these pairs indicates a value of an attribute. Since E always expresses a policy element, i.e. subject or object, its short name is predefined as either subject_type or object_type depending on the type of the relation by which E is captured as shown later in Algorithm 3, lines 8-13. However, to reach the goal of the automated or semi-automated extraction of attributes, further analysis is necessary to map attributes values represented with AV to appropriate semantic units, e.g., mapping the values senior and junior to the attribute rank. Due to the variability of values of authorization attributes, it is challenging to map attribute values to attribute names in a one-step approach. Hence, we divide this module into two major sub-modules: (1) constructing the value space of an attribute, (2) and assigning a key reference to attribute value space. The intuition is that grouping relevant values that represent one semantic unit would provide the context that facilitates predicting the respective short name. The details of each step are discussed in the following subsections.

Constructing the value space of an attribute

The list of values obtained from Module 1 is passed through this sub-module to find the mutually disjoint groups of values that constitute the value space of each potential attribute in the system. With this objective in mind, it appears that the most suitable technique to approach this problem is data clustering. For this purpose, we adopt DBSCAN algorithm, a density-based clustering method, for three reasons. First, it is difficult to predetermine the number of clusters corresponding to the number of attributes in the system without a priori knowledge of the actual requirement specifications. DBSCAN does not require the number of clusters to be known in advance. Second, clusters formed by DBSCAN can be in arbitrary shapes and therefore are more likely to be accurate clusters. Third, it has the ability to detect outliers which is the property that we need to distinguish attributes common across policy elements from those that are element specific, as discussed later in this subsection. However, the quality of the resulting clusters depends on several key design decisions. Particularly important ones are those pertaining to: (1) the specifications of the actual input data and its representation, and (2) the choice of the input parameters of DBSCAN, denoted as *eps* and *MinPts*. Next, we provide a detailed discussion regarding these concerns organized as two units of this sub-module, namely pre-processing engine and clustering component.

– **Pre-processing engine:** We seek a representation that captures the topical similarity between attribute values. The intuition is that values that constitute a value space of an attribute often occur in similar contexts, e.g. the values "senior" and "junior". Hence, we expect them to end up as neighbours, forming semantic cliques in the vector space. This in turn facilitates grouping relevant values into meaningful clusters that correspond to the attributes of the system. Words embeddings are the most reliable means to obtain such representation (Bengio et al. 2003; Collobert and Weston 2008; Mnih and Hinton 2009; Mikolov et al. 2013). Not only do word embeddings capture the contextual similarity of single-word values, but it also enables meaningful representation of multi-word attribute values using the principle of compositionality, as discussed in "Word embeddings and compositionality principle" subsection. Therefore, we utilize the embedding representation of each single-word value to encode the input to our clustering component (and the addition of the embeddings vectors in the case of multi-word input).

– **Clustering component:** given the embedding representations of attribute values, clustering methods can be used to discover cliques that constitute the value space of attributes. DBSCAN requires two parameters *eps* and *MinPts* to be tuned. Of these parameters, *MinPts* is the easier to set. Hence, the heuristic suggests tuning *MinPts* first and then *eps* can be determined according to the value of *MinPts* (Schubert et al. 2017). To be able to capture attributes with values as few as two, e.g. senior and junior as values of rank, we set MinPts to 2. If we set MinPts to be more than 2, then attributes with only two possible values would be overlooked. On the other hand, when MinPts is set to 1, each value would form a cluster on its own which is against the intuition of this step.

  To auto-tune *eps*, we should first draw the k-distance graph for all the points, where k is equal to *MinPts* (Schubert et al. 2017). Using the distance graph, we compute the average distance that separates a point from its k nearest neighbors (KNN) for all the points. Then, the average is used to set the value of the neighborhood distance cutoff, *eps*. The intuition is to establish an estimate of the distance separating core points from other points that are density-reachable (see "DBSCAN clustering algorithm" subsection) based on the density distribution obtained from the KNN.

  We run the DBSCAN algorithm, after tuning the above-mentioned parameters, with attribute values as input. The output of DBSCAN is a set of clusters. Each cluster, besides the set of outliers, represents a value space of an attribute which we want to capture, as shown in Algorithm 2 lines 11-13. Outliers, on the other hand, represent patterns with different density distributions, with respect to the value of *eps*, and therefore cannot be included in any cluster. With this explanation, we consider outliers as element-specific attribute values that have no semantic overlapping with values belonging to other elements. In this case, we collect the set of values marked as outliers by DBSCAN and that belong to same policy element according to the given element-value pairs, obtained from Module 1. Then, we repeatedly run the same process of tuning and clustering over the set of element-specific outlier attribute values (see lines 14-23 in Algorithm 2). A sample output using DBSCAN is shown in Fig. 7 with the colored clusters

```
Algorithm 2: Construct the value space of potential attributes
   Input  : @elementValuePairs: a list of element and value pairs, i.e. (E, AV), detected by relation
            R (e.g. [(nurse, first-shift), (nurse, second-shift), (nurse, junior), (lab technician,
            on-call), (lab technician, senior) etc.]
   Output : @valueSpace: a list of lists each of which corresponds to a value space of an
            attribute, e.g. [[first-shift, second-shift, on-call],[senior, junior]]
 1 Function ConstructValueSpaceOfAttributes(elementValuePairs)
 2     values = []
 3     clusteredValues = []
 4     outliers = []
 5     valueSpace = []
 6     minPits = 2
 7     for e, av in elementValuePairs do
 8         values.add(av)
 9     KNN = distanceGraph(minPits, values)
10     eps = average(KNN.distance)
11     clusteredValues = DBSCAN(minPits, eps, values)
12     outliers = clusteredValues.getOutliers()
13     valueSpace.add(clusteredValues\outliers)
       // returns a dictionary that maps
          an element with the list of
          values that it has in outliers
14     outliersElm= groupValuesByElement(outliers, elementValuePairs)
15     for e, values in outliersElm do
16         do
17             KNN = distanceGraph(minPits,values)
18             eps = average(KNN.distance)
19             clusteredValues = DBSCAN(minPits, eps, values)
20             outliers = clusteredValues.getOutliers()
21             valueSpace.add(clusteredValues\outliers)
22             values = outliers
23         while size(values) > minPts;
24     return valueSpace
```

representing value space of two attributes potentially named "working_hours" (the red cluster) and "rank" (the blue cluster).

Assigning a key reference to the value space of an attribute

Each cluster obtained from the previous step represents a value space of an attribute. At this point, we aim at determining a suitable attribute name for each cluster. Bakhshandeh et al. have suggested using nouns in attribute value glosses or definitions to guide this naming process (Bakhshandeh and Allen 2015). Following their approach, we propose using nouns common across the definitions of all values in a cluster to generate a set of candidate attribute names. For instance, using WordNet (Miller 1995), the definitions of senior and junior are:

1. Senior: older; higher in rank; longer in length of tenure or service
2. Junior: younger; lower in rank; shorter in length of tenure or service

Thus, rank, length, tenure and service are all possible names. To assign the actual short name, we assume a human-in-the-loop scenario in which the human analyst (i.e. security architect) selects or custom-defines the most appropriate attribute name. It is worth mentioning that the functionality of the subsequent steps is not affected by the chosen name as long as it is unique for each cluster, which is the default setting in our experiment.

Once the key is assigned to a value space, line 4 in Algorithm 3, further processing is needed to group values within each value space by the element to which they belong. Then, it proceeds to map the attribute key/value pairs to their respective element (see Algorithm 3) lines 5-7. This order of execution allows defining a common attribute name across different policy elements which in turns facilitates inheritance analysis as discussed in the next module. In the remaining of the Algorithm 3, lines 8-13, policy elements are assigned as values of the predefined attributes "subject_type" or "object_type" of the predefined element "subject" or "object".

Module 3: Assigning attribute to namespaces hierarchically

Namespace is the subject or object group to which an attribute belongs. It can trivially be specified as one-to-one mapping with policy elements. Consider, for instance, our running example (see "Running example" subsection).

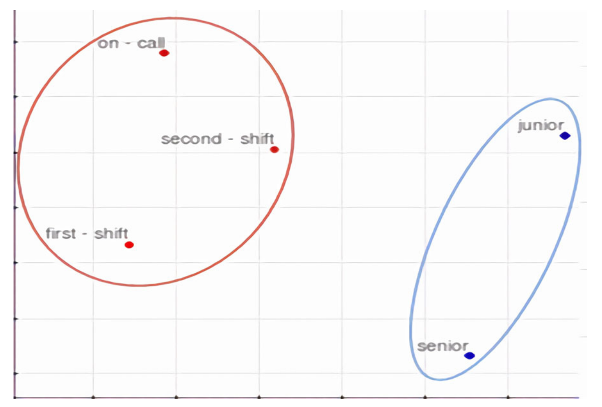

Fig. 7 Example of clusters obtained with DBSCAN. Red cluster represents a value space of an attribute potentially named working_hours while the blue cluster contains value of an attribute potentially named rank

Algorithm 3: Assign a key reference to each value space

```
Input  : @R: the relation e.g. subject relation.
         @elementValuePairs: the list of element and
         value pairs, i.e. (E, AV), detected by relation
         R. @valueSpaces the output of Algorithm 2
Output: @keysValues: a list of triples where each
         triple is represented as (key, values, policy
         element), e.g. [(working_hours, [on-call],
         lab technician), (working_hours,
         [first-shift, second-shift, on-call ], nurse)]
1  Function suggestShortName(R, elementValuePairs, valueSpaces)
2      elements = []
3      for vs in valueSpaces do
           // assigns a short name for each cluster as discussed in
              "Assigning a key reference to the value space of an
              attribute" subsection
4          key = getKeyReference(vs)
           // returns a dictionary that maps an element with the
              list of values that it has in vs
5          vsElm= groupValuesByElement(vs, elementValuePairs)
           // assigns key and value pairs to the element to which it
              belongs
6          for e, values in vsElm do
7              keysValues.append((key, values, e))
       // adds policy elements as values to the predefined key, e.g.
          subject_type, and assigns it to the predefined group, e.g
          subject, according to the relation type.
8      for e, av in elementValuePairs do
9          elements.add(e)
10     if R.type() is subject-attribute then
11         keysValues.append(("subject_type", elements, "subject"))
12     if R.type() is object-attribute then
13         keysValues.append(("object_type", elements, "object"))
14     return keysValues
```

Grouping the attributes obtained from Module 2 by policy elements would result in a total of five namespaces (subject groups), namely:

1. subject
   - {subject_type: nurse, lab technician, employee, patient }
2. nurse
   - {rank: senior, junior}
   - {working_hours: on-call, first-shift, second-shift}
3. lab technician
   - {rank: senior}
   - {working_hours: on-call}
4. employee
   - {rank : senior}
5. patient
   - {status : registered}

However, this method yields a flat representation of ABAC system which contains several redundancies of attributes across namespaces. On the other hand, defining a hierarchical structure among these groups uncovers redundant groups and allows multiple attributes to be assigned with a single attribute assignment. This reduces the overall cost required to tag similar elements with a common attribute and value pairs (Servos and Osborn 2015). Inspired by the work of (Stoller and Bui 2016), we define a two-steps heuristic approach to capture the implicit hierarchical structure among namespaces given the flat representation. These steps are to (1) compute inheritance and (2) compute assigned attributes. The corresponding details are discussed in the rest of this subsection.

- **Compute inheritance:** The goal is to determine inheritance relationships between namespaces. An ABAC model is said to meet the requirements of full inheritance if the inheritance relation holds between every two namespaces in the system. Guo et al. have referred to this property as completeness in the context of RBAC (Guo et al. 2008). On the basis of the property of full inheritance, we define an isAncestor(ns_1, ns_2) function. This function compares two namespaces ns_1 and ns_2 with respect to their attributes; ns_1 is an ancestor of ns_2 if the attributes of ns_1 is a subset of the attributes of ns_2. The intuition is to define a parent-child relation between elements, i.e., namespace, such that a parent namespace contains attributes common between its children. Here, attributes comparison is done according to their short names and values. For strict inheritance, attributes a_1 and a_2 are considered equivalent if they have identical name and value set. On the other hand, under simple inheritance, a_1 and a_2 are considered as

Algorithm 4: Compute inheritance between elements

```
Input  : @inheritanceMode: a flag that specifies the
         inheritance mode as strict or simple.
         @mainNsID: the name of the predefined
         super group, i.e. subject or object.
         @keysValues: the output of Algorithm 3.
Output: @parents: dictionary that maps each
         namespace to its parents. @consNs: a list of
         consolidated subject/object groups. Each
         group has a name and is assigned with the
         attributes key/value pair
1  Function ComputeInheritance (inheritanceMode,
   mainNsID, keysValues)
   // returns a list of namespaces
      representing the flat represe-
      ntation of ABAC. Each namespace
      has a name and is associated with
      set of attributes.
2  flatNamespaces =
   groupAttributesByElement(keysValues)
   // returns the main namespace i.e.
      subject or object
3  mainNamespace = getElement(flatNamespaces,
   mainNsID)
4  parents =new Dictionary()
5  equivalents =new Dictionary()
6  for ns in flatNamespaces do
7      parents(ns) = new Set()
8      equivalents(ns) = new Set()
9  for $ns_i$ in flatNamespaces do
10     for $ns_j$ in flatNamespaces \ $ns_i$ do
11         if equals($ns_i$, $ns_j$, inheritanceMode) then
12             equivalents($ns_i$).add($ns_i$)
13 consNs = consolidate(equivalents)
14 for $ns_i$ in consNs do
15     for $ns_j$ in consNs \ $ns_i$ do
16         if isAncestor($ns_j$, $ns_i$, inheritanceMode) then
               // checks if $ns_j$ is a parent
                  or an ancestor of $ns_i$
17             if ! $\exists$ $ns_k$ in parents($ns_i$) s.t. isAncestor
               ($ns_j$, $ns_k$, inheritanceMode) then
18                 parents($ns_i$).add($ns_j$)
19                 for $ns_k$ in parents($ns_i$)\$ns_j$ do
                       // checks if $ns_i$ has a
                          parent that is also
                          an ancestor of $ns_j$
20                     if isAncestor
                       ($ns_k$, $ns_j$, inheritanceMode) then
21                         parents($ns_i$).remove($ns_k$)
22     if parents($ns_i$) is empty then
23         parents($ns_i$).add(mainNamespace)
24 return parents, consNs
```

equivalent if they have a common name and overlapping value set. To generate a graph-like ABAC structure, each namespace is mapped to its parents using the function, isAncestor, as shown in Algorithm 4 lines 16-25. The predefined element , i.e. "subject" (or "object" in the case of object groups) is referred to as main namespace and is assigned as a parent of namespaces that have no parents , lines 24-25 in Algorithm 4. All other elements —e.g., nurse, lab technician, employee etc. in this example— are descendants of this element. In the special case where two elements are equivalent, i.e., have similar sets of attributes, the elements are consolidated on one namepace, lines 8-12 Algorithm 4. The assigned name of the new namepace is the least common Hypernym[2] of the two underlying elements, obtained using WordNet (Miller 1995).
Under the assumption of simple inheritance, "nurse" and "lab technician" are examples of this special case as they represent two distinct subject groups with the equivalent set of attributes. After consolidation, these two groups are merged into one logical unit referred to as "health_professional" and to which the attributes of the redundant groups are assigned.

– **Compute assigned attributes:** This step computes the directly assigned attributes of each namespace by removing the attributes inherited from its parent as shown in Algorithm 5. An illustration of subject group hierarchy under the assumption of simple inheritance is shown in Fig. 8.

Module 4: Identifying the category of an attribute

A straightforward method to determine the attribute category is to refer to the type of relation by which the attribute is detected. The categories subject and object

Algorithm 5: Compute directly assigned attributes

```
Input  : @parents and @consNs as obtained from
         Algorithm 4
Output: @namespace: a list of subject or object
         groups each with the directly assigned
         attributes
1  Function ComputeAssignedAttributes (parents,
   consNs)
2      namespaces = []
3      for ns in consNs do
4          inheritedAtt = $\cup_{parents(ns).attributes}$
5          assignedAtt =
           ns.attributes.removeAll(inheritedAtt)
6          namespaces.add(ns.name, assignedAtt)
7      return namespaces
```

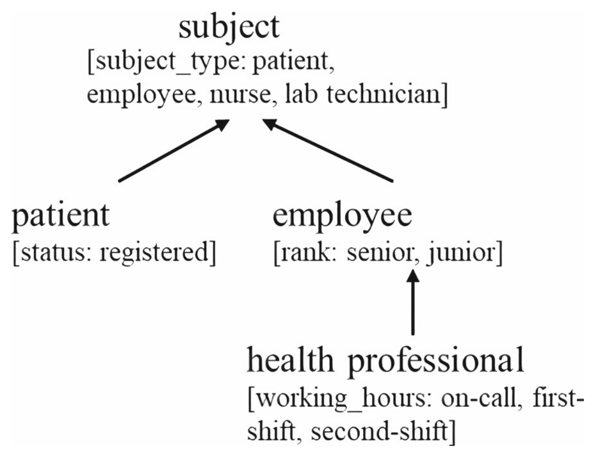
subject
[subject_type: patient, employee, nurse, lab technician]
patient
[status: registered]
employee
[rank: senior, junior]
health professional
[working_hours: on-call, first-shift, second-shift]

Fig. 8 Example of subject group hierarchy represented as a graph. Below each group name is the list of directly assigned attributes

correspond the relations subject-attribute and object-attribute, respectively.

Module 5: Identifying an attribute's data type

We employ a heuristics-based approach to infer the data type of an attribute. The heuristics are direct mapping from each NE type to a data type. Consider, for instance, the policy sentence: "$A\underline{nurse}_{(concept)} at the \underline{MayoClinic}_{(organization)} can view the relevant \underline{labtests}_{(concept)}$." A named entity tagger (Dandelion api 2018) assigns the type of each entity as shown in the subscripts . In this example, "concept" and "organization" are mapped to string. A limitation of this approach is that it can infer a data type of an attribute only if NE tagger successfully captures the NE type of its value, i.e. E or AV.

Once all information pertaining the five properties of attribute are collected, new attributes can be defined as shown in Algorithm 6.

Algorithm 6: Define attributes of the hierarchical ABAC system

```
Input  : @R: relation e.g. subject-attribute.
         @parents: the output of Algoritm 4,
         @namespaces: the output of Algoritm 5.
Output: List of attributes defined with the short
         name, namespace, category, data type and
         values as illustrated in Table 1
1  Function addNewAttributes (R, parents, namespaces)
2      category = R.type()
3      for ns in in namespaces do
4          for att in ns.attributes() do
5              shortName = att.key()
6              values = att.values()
7              nsName = getHierarchicalPath(att, ns, parents)
8              dataType = getDataType(values)
9              attributes.add(shortName, nsName, category, dataType, values)
10     return attributes
```

Corpus creation

While the advancements of ML and NLP techniques can provide ABAC security architects with powerful tools to address requirements analysis and attributes extraction tasks, the effective adoption of these techniques is restricted by the lack of appropriate datasets. One reason for this dearth of data is that the requirement specifications in general, and authorization requirements in particular are meant to be proprietary except for a few software projects for which requirements are publicly available. However, the majority of available requirements documents often express authorization rules by the means of roles, actions and resources, e.g., "a patient chooses to view his or her access log." While these elements are considered as attributes, they mainly capture the core policy elements. Hence, these are not enough to train an automated attribute extraction model. To remedy the lack of appropriate data, we construct a synthetic natural language policy dataset such that the authorization rules are expressed with attributes. We then use this dataset to evaluate the proposed approach.

Two information sources have been used to fuel the synthetic data generation framework. The first is the datasets obtained from prior efforts in natural language access control policy (NLACP) collection. These datasets were constructed to study various aspects of the automated translation of NLACPs to machine-readable form. The most representative examples of these datasets are :

1. iTrust: is a patient-centric application for maintaining an electronic health records (Meneely et al. 2011).
2. IBM Course Management App: is a registration system for colleges (IBM 2004).
3. CyberChair: is a conference management system (Van De Stadt 2012).
4. Collected ACP: is a dataset consists of combined access control policy sentences collected by Xiao et al. (2012).

Herein we refer to these documents collectively as ACP dataset[3]. The authoring style of the authorization requirements in the ACP dataset is tuned more towards the role-based access control than attribute-based access control. Hence, these datasets do not have enough attributes for the purpose of our study. To address this limitation, we augment either the subjects or objects elements of the policies with additional descriptive context derived from a general purpose textual data, namely Common Crawl

(Common crawl 2018), which composes our second information source. The purpose of injecting this additional context is to change the tone of the policy so it reads as if it was written with ABAC model in mind. For example, before injecting the descriptive context, a policy sentence might be written as "faculty can access library materials", while after injection, it becomes "*retired USD* faculty can access the *unrestricted* library materials." The latter sentence ties together the core policy elements with their attributes, i.e. *USD*, *retired* and *unrestricted*, to mimic the type of authorization requirements ABAC model is meant to capture.

Figure 9 shows the overall synthetic data generation framework. We start the process with the subject and object elements of access control sentences in ACP dataset as they were identified by the original authors. Next, we search through the Common Crawl web content to retrieve segments of text that contain the elements of interest, i.e. the subject and the object. We limit the language of the retrieved segments to English. Intuitively, not all of the matching segments appropriately fit with the semantic of policies in ACP dataset. To filter irrelevant matches, we first build a language model using the ACP dataset. Then, we use the conditional probability of each segment under the computed language model to determine the semantic fitness of a segment to the context of ACP dataset. Next, we manually augment policy sentence with the relevant portion of text from the matching segment and make necessary changes. The injection process ends once we collect as much of the relevant segments as needed to inject sentences of ACP dataset with the synthetic attributes.

One might argue that the data generation process can be done entirely manually, by recruiting individuals to do so. An obvious motivation for the semi-automation is to generate the synthesized policies as natural as can be. This is to say that data generated manually by individuals in a controlled experiment settings might be biased by the individuals' writing styles, their background knowledge as well as their comprehension of the data generation instructions. However, using content drawn from a general purpose content repository, such as Common Crawl, captures the natural contexts that normally contains the elements of interests. Scalability, in terms of the volume of data and the variety of domains, is another motivation. The reason is that an automated means of data generation is designed to at least partially eliminate the need of domain experts. Table 2 shows examples of the synthesized policies which if generated manually would require a certain level of domain-specific knowledge.

To this end, we introduce our synthetic natural language policy dataset of four different domains. It contains a total of 851 sentences with manual annotations of 867 subject-attribute and 932 object-attribute instances. Table 3 summarizes the number of elements along with their attributes in our dataset. To capture the subject and object elements of ABAC policy along with their attributes, we develop the following annotation scheme:

1. e2SA: stands for element#2 in Subject Attributes Relation. A segment of the sentence that is tagged with this label corresponds to a value of a core policy element named subjectType
2. e1SA: stands for element#1 in Subject Attributes Relation. This label is used to tag the part of the sentence that expresses a value of an attribute that modifies e2SA

We similarly define e2OA, e1OA to label objects and object attributes, respectively. Listing 1 presents the format for an annotated ACP sentence.

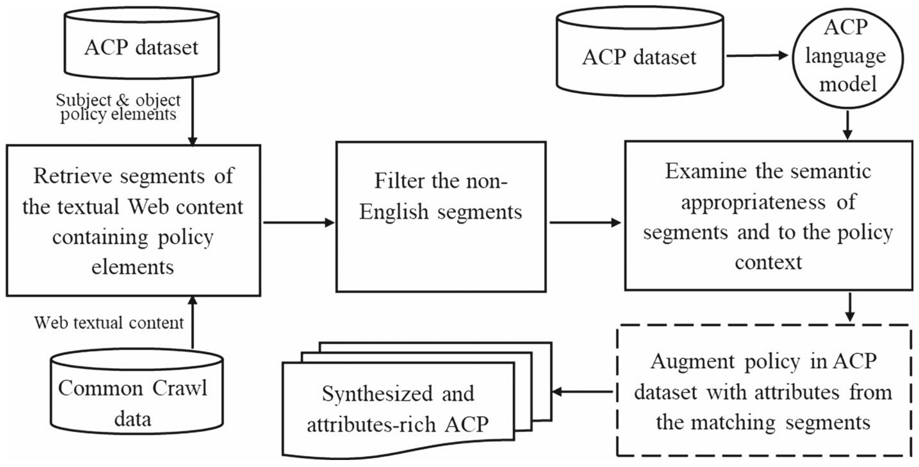
ACP dataset
Subject & object policy elements
Retrieve segments of the textual Web content containing policy elements
Web textual content
Common Crawl data
Filter the non-English segments
ACP dataset
ACP language model
Examine the semantic appropriateness of segments and to the policy context
Augment policy in ACP dataset with attributes from the matching segments
Synthesized and attributes-rich ACP

Fig. 9 Synthetic data generation framework. The dashed box indicates the manual task in this process

Table 2 Examples of the synthesized natural language policies

A	B
A lab technician can update the status ...	A full-time first shift lab technician can update the status ...
Professor must be able to access the system to ...	Professor, of economics at University of NSW, must be able to access the system ...
Program Committee (PC) must be selected to review the papers.	Program Committee (PC) must be selected to review the "borderline" papers.

Column A contains the original policies as obtained from ACP datasets, and B shows the policies after being augmented with attributes

Experimental results and performance evaluation

We next present the evaluation we conducted to assess the effectiveness of our approach in extracting attribute values and in constructing the value space of potential attributes of each policy element, i.e., "Module 1: Attribute extraction" and "Module 2: Suggesting attributes short names" subsections, respectively. In our evaluation, we focus on two main research questions:

- **RQ1:** How effectively —in terms of precision, recall and F_1-score— does our approach extract attributes of subjects and objects?
- **RQ2:** How effectively —in terms of precision, recall and F_1-score— does our approach construct the value space of potential attributes?

To answer RQ1, we first generate the dependency tree representation of each sentence. We augment the resulted representation with annotations needed to explore patterns encoding either subject-attribute or object-attribute relations. Our analysis shows that there exist 125 unique patterns encoding subject-attribute relation, while the object-attribute relation is encoded with 240. Table 4 shows the five most frequently occurring patterns for each relation as per our corpus. The remaining patterns account for very few valid instances individually. Thus, we only consider the 5 most occurring ones. Matching against these 5 patterns generates two sets of candidate instances, one for each relation. It is worth mentioning that identifying these patterns is not only valuable to the purpose of this study, but also to collect seeds to establish an inductive approach to semi-automate the process of learning patterns of our target relations (Jurafsky and Martin 2009).

Table 3 The corpus built for the experiment

Dataset	ACP sentences	# Subject-attribute	# Object-attribute
iTrust	466	511	534
Collected ACP	112	144	100
IBM App	163	140	195
CyberChair	110	72	103
Total	851	867	932

Listing 1 Annotated ACP sentence

```
An [on−call#e1SA][senior#e1SA][nurse#
    e2SA] may change the list of [
    approved#e1OA] [lab procedures#e2OA]
```

Next, we build our CNN model with a total of 4 layers of which two are convolution layers with dropout of 0.2 followed by one max pooling layer and the sigmoid layer. Throughout the experiments, we utilize the publicly available GloVe pretrained word embeddings of 300 dimensions. GloVe embeddings are trained on 42 billion words of Common Crawl data using the global word-word co-occurrence statistics (Pennington et al. 2014, 2017). For the convolutional layer, we use Relu as an activation function and 128 filters with a window of size 2. We use default values for other parameters, e.g., batch size and kernel initializer, are left as default. The models (i.e. subject-attribute and object-attribute) are trained for 10 epochs. To evaluate the performance of our CNN, we employ a 3-fold stratified cross-validation as well as document fold validation. In the document fold, the models are trained on three documents,i.e., datasets, and tested on the sample of the fourth. The rationale of such evaluation setting is to evaluate how the system will perform on out-of-domain data. The evaluation measures are recall, precision and F1-score as shown in Table 5 and defined as follows:

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$F_1 = 2\frac{Precision \times Recall}{Precision + Recall} \tag{4}$$

Where *TP*, *FP* and *FN* are the true positives, false positives and false negatives which are discussed in Sokolova and Lapalme (2009).

Table 4 Five most occurring patterns in each relation (for more information regarding components of each pattern, e.g., nsubj, see De Marneffe and Manning (2008))

Subject-attribute	Object-attribute
nsubj,amod	dobj,amod
nsubj,prep	pobj,amod
nsubjpass,amod	dobj,prep
nsubj,compound	dobj,compound
nsubj,ROOT,amod	nsubjpass,amod

Table 5 Performance results of extracting attribute values using subject-attribute and object-attribute models (P, R, F_1 represent precision, recall and F_1-score, respectively)

Dataset	Subject-attribute			Object-attribute		
	P	R	F_1	P	R	F_1
iTrust	0.97	0.99	0.98	0.91	0.90	0.90
Collected ACP	0.93	0.93	0.93	0.93	0.95	0.94
IBM App	0.95	0.96	0.96	0.92	0.92	0.92
CyberChair	0.97	0.98	0.97	0.88	0.87	0.88
Document fold	0.91	0.80	0.85	0.75	0.69	0.71

The first four rows show the performance of the model when tested on in-domain data whilst the last row represents how the system would perform on out-of-domain data

A general observation with regard to the performance behavior is that the subject-attribute relation extraction model surpasses the performance of object-attribute model. A possible interpretation of the overall strength of the subject-attribute relation (see Table 5) is rooted in the nature of the instances of this relation. More specifically, the subject-attribute relation is meant to link system's users, which are mostly human entities, with their corresponding attributes. This introduces a level of semantic consistency between instances belonging to this particular relation. However, this is not the case with the object-attribute relation due to the high variability of its instances. This makes it less likely to be learned from a relatively small data size. We, therefore, expect that the accumulation of more labeled instances can substantially improve the performance.

The first four rows of Table 5 show the means of the 3-fold cross validation across 10 runs. Scores represent the performance of the model in detecting the valid instances of both relations, i.e. the performance over the positive class. The variance of these means ranges between 0.01 to 0.1. This experiment, the n-fold, represents the performance of the model on in-domain data. The high score of the precision and recall indicates model's ability to infer the semantic consistency among candidate instances. To evaluate how the model would perform when tested on out-of-domain data, we conduct a document fold experiment. Here, the model is trained on 3 documents and tested with instances that belong to the fourth domain. The results of this experiment have shown an expected drop in the values of precision and recall. However, the loss is more significant in the case of object-attribute relation. A potential justification for this behaviour is that despite the domain differences, there are common attributes of subjects, e.g. location, age .. etc. that are shared across domains. Such similarity is less likely to occur between objects across domains. For instance, a model trained on data from a medical domain would not be certain about the attributes of, say, courses and class as does with medical records. It is also worthy to note that we have experimented with several other classical learning algorithms, e.g. SVM and Naive Bayes, trained on manually engineered features; but, the CNN outperforms other models.

In addition, we examined how would different percentage of labeled data affect the performance of the attributes extraction model. As shown in Fig. 10, the performance of subject-attribute relation begins in the range 0f 0.60 to 0.77 F1-score with just 25% of the labeled instances per dataset. Then, the performance on the four datasets improves as the amount of available data increases. The same observation holds for object-attribute relation. Due to the space limit, we only show the results of scalability test on subject-attribute relation (see Fig. 10).

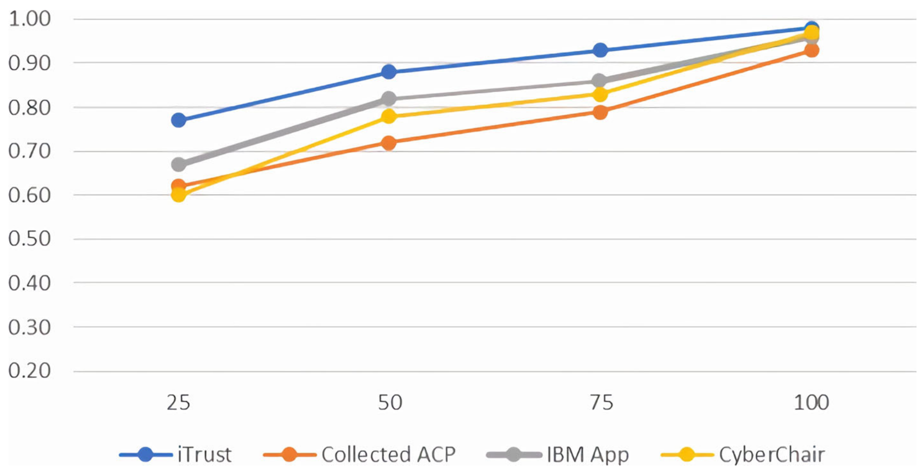

Fig. 10 Scalability test results of the subject-attribute relation.The x-axis indicates the percentage of data used during the experiment while the y-axis shows the performance using the F_1 score

In order to address the question RQ2 pertaining the performance of our approach in constructing the value space of subject and object attributes, we run Algorithms 2 and 3 as discussed in "Module 2: Suggesting attributes short names" subsection. Using our dataset, we repeated this experiment twice, once with the values of subject attributes as input and the other with the values of object attributes. The count of the attribute values is shown in Table 3. We again utilized the GloVe pretrained word embeddings of 300 dimensions for transforming the textual attribute values to numerical vectors. This experiment resulted in clusters of values as exemplified in the rows of the last column in Table 7. For evaluation, the corresponding class label of each cluster (i.e., the short name) is determined according to the concept that makes up the majority of values in a cluster. Then, the goodness of each cluster is measured using the notion of recall, precision and F_1-score. In the context of clustering, recall, precision and F_1-score of cluster j with class label i are defined in Karol and Mangat (2013); Steinbach et al. (2000) as follows:

$$Recall(i,j) = \frac{n_{ij}}{n_i} \tag{5}$$

$$Precision(i,j) = \frac{n_{ij}}{n_j} \tag{6}$$

$$F_1(i,j) = 2\frac{Precision(i,j) \times Recall(i,j)}{Precision(i,j) + Recall(i,j)} \tag{7}$$

Where n_j is the number of attribute values in cluster j, n_i is the number of values in class i and n_{ij} is the number of attribute values belonging to class i in cluster j. Table 6 shows the average precision, recall and F_1-score across resultant clusters that represent value spaces of attributes in our data. In the majority of cases, the performance is promising.

To demonstrate the capabilities of our approach, Table 7 presents examples of clusters detected in our experiment. As can be seen, our approach successfully defines reasonable attributes of various policy elements by clustering relevant values into value spaces. For instance, we were able to automatically capture five attributes of a professor and four for a lab technician and the same applies on other elements in Table 7. To further understand the success and failure factors that affect the quality of clusters,

Table 6 Performance results of constructing the value space of subject and object attributes

Dataset	Subject attributes			Object attributes		
	P	R	F	P	R	F
iTrust	0.82	0.84	0.81	0.80	0.79	0.78
Collected ACP	0.89	0.90	0.89	0.82	0.85	0.82
IBM App	0.89	0.87	0.88	0.88	0.86	0.85
CyberChair	0.85	0.87	0.85	0.79	0.85	0.80

Table 7 Sample results of attributes key/value pairs of several subject elements in our dataset

Policy element (dataset)	Attribute	
	Short name/cluster label	Value space
Professor (IBM App)	Employment_type	Full-time, part-time
	Track	Tenure, non-tenure
	Department	Of economics, department of informatics, at the college of information sciences, at the department of finance
	College	At the business school
	University	At the university of newcastle, At the university of nsw
Lab technician (iTrust)	Rank	Senior, junior
	Affiliation	At the cancer research lab
	Working_hours	On - call, first - shift, second - shift
	Qualification	With a bachelor's degree, with IHC certificate
Administrator (iTrust)	Rank	Senior, junior, associate
	Affiliation	General hospital, children's Hospital weman's hospital
	Authorization	Licensed, unlicensed
Author (CyberChair)	Type	Corresponding, contributing

The value space of each attribute is detected as discussed in the "Module 2: Suggesting attributes short names" subsection

we performed a manual inspection of the experimental results. Building upon this analysis, we made two main observations. First, forming quality clusters requires complete requirements specification. Note that the idea of using clustering techniques to construct value spaces of attributes is built upon the assumption of the presence of related values. This is a fair assumption since, for example, defining an authorization requirement that targets, say, "senior lab technicians" implies that different considerations are made for the "juniors". The presence of both attribute values, i.e., senior and junior, in the NLACPs enables the discovery of meaningful clusters and vice versa. Second, a general purpose word embeddings, e.g., embeddings trained on Common Crawl data, might be inadequate in capturing the semantics of domain sensitive attribute values. The fact that could justify the performance drop when experimenting with object attributes of iTrust and CyberChair datasets. Consider for instance the following sentences from CyberChair document:

- Program Committee (PC) must be selected to review the **borderline** papers.
- Program Committee Chair (PCC) must inform the maintainer about the **rejected** papers.
- The corresponding authors of **accepted** papers must submit the brushed-up versions of the papers.

Unlike the words "accepted" and "rejected", that have a more or less a consistent semantic regardless the context, the semantic of the word "borderline" is context dependant and domain sensitive. In such a case, a general purpose embeddings, similar to the one we used, might not capture the relatedness among the words (i.e., the attribute values). One technique to address this issue is to utilize a domain-specialized embedding representation of words as discussed in Dingwall and Potts (2018).

It is worth mentioning that, aside from the DBSCAN algorithm, we have also explored the use of K-means (Hartigan and Wong 1979) and affinity propagation (AP) (Frey and Dueck 2007) as representative techniques of the partition-based clustering and exemplar-based clustering, respectively. However, for our purpose DBSCAN performs better than other two the algorithms.

Limitations

Evaluating the proposed approach with real-world requirements would be ideal. Hence, the use of the synthetic data might impose a limitation on this work. However, acquiring the real data from real organizations might not be feasible considering the privacy-sensitive nature of the data. We, therefore, develop our synthetic data that, while not real, is intended to be realistic. Particularly, to ensure that our synthetic data preserves the key characteristics of real data, we made the following design decisions: (1) the data was built upon real requirement documents, (2) the injected attributes were obtained from the Web content in an attempt to capture the real-world context of a particular policy element, and (3) the manual part of the injection process was conducted by Ph.D. students with expertise in access control domain. Hence, we believe that our synthetic data is representative of the real-world data and so is the performance. Generally speaking, the nature of privacy-sensitive content is a major challenge in devising data-driven security/privacy solutions, such as insider threat detection or clinically-relevant text analysis tools, to name few. In such situations the synthetic content has been considered as an alternative (Glasser and Lindauer 2013; Oak et al. 2016). Another limitation of the current work is that it captures the attributes if their values are explicitly mentioned in the NLACPs. However, situations exist where authorization attributes are implicitly mentioned and need to be understood from the context. This is likely to occur when a pair of subject and object attributes need to match for a rule to be applicable. Consider for example the following sentence:

- To be able to protect the reviews from outsiders, each **reviewer gets his or her own directory**

The bold portion of the sentence implies the presence of two attributes, namely ID and owner, that are required to assert the condition; reviewer.ID == directory.owner. Since the values of these attributes are not explicitly expressed in the sentence, the current system cannot recognize them.

Related work

We categorize the related work into two different research lines: (1) automated or semi-automated transformation of natural language access control policy to machine-executable form (referred to as top-down policy engineering) and (2) policy mining (known as the bottom-up approach of policy engineering).

Natural language and access control policy

Several studies researched various aspects of the automated analysis of natural language ACPs. In order to review these prior efforts, we introduce six comparison dimensions along which each related work has been characterized: (1) the building blocks of the proposed framework; (2) the underlying techniques used for each component; (3) indicators or features used if learning approach is employed; (4) the dataset used for the evaluation; (5) size of the dataset (measured by the number of sentences), and (6) the average of available performance metrics. Table 8 presents a summary of the findings, while the rest of this section discusses each study separately.

Xiao et al. (2012) were the first to introduced an automated approach, named Text2- Policy, to construct access control policy rules out of natural language project artifacts. They built their approach upon matching against 4 predefined patterns to discern ACP sentences from other types of sentences. Then, they defined heuristics based on the same set of patterns to extract instances that jointly compose ACP rules. The extraction task is followed by a transformation process in which a formal representation of the extracted rule is generated. The reported results achieved an average recall of 89.4% with a precision of 88.7% in the identification phase and an accuracy of 86.3% on the extraction. Evaluation results of the transformation phase were not reported. However, their approach cannot capture ACP sentences that do not follow the predefined patterns; and, it has been shown that only 34.4% of ACPs

Table 8 ACP extraction from natural language artifacts in the related work

Study	Proposed framework	Underlying tech	Indicators	Dataset	Size	Performance
(Xiao et al. 2012)	ACP sentence identification	Semantic patterns matching	N/A	iTrust, IBMApp	927	Prec:88.7% Rec:89.4%
	ACP elements extraction	Heuristics over the patterns	N/A	Access control (AC) sentences in: iTrust, IBM App, and collected ACP	241	Accu:86.3%
	Transformation to formal model	Heuristics	N/A	N/R	N/R	N/R
(Slankas and Williams 2013)	ACP sentence identification	Majority vote of KNN, Naive Bayes (NB) and SVM classifiers	Words,synonyms,POS, named entities, & Levenshtein distance in the case of KNN	iTrust	1159	Prec:87.3% Rec:90.8%
	ACP elements extraction	RE using bootstrapping; seeding patterns are derived from dependency tree	N/A	AC sentences in: iTrust	409	Prec:46.3% Rec:53.6%
		NB classifier of candidate instances	were not clearly reported			
(Slankas et al. 2014)	ACP sentence identification	KNN	Levenshtein distance	iTrust,IBM App, Cyberchair, collected ACP	2477	Prec:81% Rec:65%
	ACP elements extraction	RE using bootstrapping; seeding patterns are derived from dependency tree	N/A	AC sentences in: iTrust,IBM App, Cyberchair, collected ACP	1390	N/R
		NB classifier of candidate instances	Pattern itself, relationships to resource and subject, POS of subject and resource			
(Narouei et al. 2017)	ACP sentence identification	NB and SVM classifiers	A total of 821 features categorized into: pointwise mutual information, security,syntactic complexity, and dependency features	Trust, IBM App, Cyberchair, collected ACP	2477	Prec:90% Rec:90%
(Narouei et al. 2017)	ACP sentence identification	Deep recurrent neural network	Words embeddings	ACPData, iTrust,IBM App, Cyberchair, collected ACP	5137	Prec:81.28% Rec:74.21%
(Narouei and Takabi 2015a)	ACP elements extraction	SRL	N/A	AC sentences in: iTrust, IBM App, Cyberchair, collected ACP	726	Prec:58.3% Rec:86.3%
(Narouei and Takabi 2015)	ACP elements extraction	SRL	N/A	AC sentences in: iTrust, IBM App, Cyberchair, collected ACP	841	Prec:63.5% Rec:86.25%

In this table, N/A stands for not applicable while N/R means not reported can be captured with Text2Policy's patterns (Slankas et al. 2014).

Slankas et al. (Slankas and Williams 2013) proposed Access Control Relation Extraction (ACRE), a machine learning based approach for ACP elements extraction from natural language documents. ACRE can be broken down to an ACP sentence identification phase followed by ACP elements extraction. In the identification phase, the authors investigated whether words, words' synonyms, POS and named entities could be used as indicators to identify ACP sentences. In the elements extraction phase, a bootstrapping technique built upon patterns drawn from dependency tree representations of sentences has been adopted to extract ACP instances. Slankas et al. empirically validated the proposed approach on a version of iTrust that contains 1159 sentences. Their approach achieved a recall of 90.8% with a precision 87.3% in the identification phase, whereas in ACP rule extraction authors reported 46.3% precision with a recall of 53.6% as the best performance they can achieve.

Slankas et al. (2014) extended ACRE, which was first introduced in Slankas and Williams (2013). The main components of the framework, as well as the underlying techniques, are similar to their original proposal with only subtle modifications. What clearly distinguishes this work from its predecessor is the evaluation. In Slankas et al. (2014), Slankas et al. validated their proposed approach against larger datasets collected from 5 sources of policy data that were previously used in the literature. For the identification phase, K-nearest neighbor (KNN) learning approach was employed to discern ACP sentence from other types of sentences. Further, the authors achieved an average classification precision and recall of 81% and 65%, respectively. For elements extraction phase, on the other hand, performance was reported per dataset, meaning the overall average scores were not reported.

Narouei et al. (2017) designed a new set of features to distinguish ACP from non-ACP sentences. Using the proposed feature set and following the classical machine learning pipeline, the authors reported an average precision and recall of 90%.

Narouei et al. (2017) proposed a top-down policy engineering framework to particularly aid ABAC policy authoring. They adopted the following pipeline: ACP sentence identification followed by policy elements extraction phase. For the identification task, the authors used a deep recurrent neural network that uses pre-trained word embeddings to identify sentences that contain access control policy content; and, they achieved an average precision of 81.28% and 74.21% for recall. For policy elements extraction, the authors suggested the use of semantic role labeler (SRL), but no evaluation results were reported (Narouei et al. 2017). Using the language of ABAC model, the authors regarded the arguments of SRL tagger as the attribute values while argument definitions were seen as the keys or what are so called short names. Consider for instance the two ACPs sentences : (1) A registered patient can access his/her full health record and (2) A registered patient can read his/her full health record. Following the approach of Narouei et al. (2017), attributes would be represented as follows:

– accessor = patient
– reader = patient

where *reader* and *accessor* are the arguments definition obtained from the SRL tagger. Such a direct mapping from the SRL output (i.e. arguments and argument definitions) to attribute values and short names of ABAC model is insufficient. This is mainly because of two reasons. First, SRL labels arguments only with respect to the predicate, e.g. *access* and *read*. Therefore, attributes of subjects and objects such as *registered* and *full* will not be captured using this approach. Second, from the SRL perspective, an argument definition, e.g. *accessor* and *reader*, describes the semantic relation between an argument and a predicate in a sentence. This description, however, might not be an appropriate short name of an attribute as can be seen in the above-mentioned example. That is from ABAC standpoint, *patient* is a value of one attribute called subjectType, or any semantically similar attribute name, rather than the two attribute names —*reader* and *accessor*— suggested by the SRL tagger.

Narouei and Takabi (2015a, b) proposed a new approach for extracting policy elementls from ACP policy sentences using semantic role labeler (SRL). The authors reported that they were able to achieve higher extraction recall (88%) when compared to ACRE (Slankas et al. 2014). This boost of recall, however, comes at the cost of precision. While we are aware that the lack of benchmark dataset makes the comparison task non-trivial, the reported results suggest that a combination of policy extraction approaches by Slankas et al. (2014) and Narouei and Takabi (2015a) can potentially balance both precision and recall values. Further, Narouei et al. (2018) investigated the idea of improving SRL performance using domain adaptation techniques. The authors reported that they were able to identify ACP elements with an average F1 score of 75%, which bested the previous work by 15%.

Turner designed an ABAC authoring assistance tool that allows a security architect to configure an ABAC expression as a sequence of fields, namely subject, subject attributes, object, and object attribute, etc. (Turner 2017). The natural language ABAC policy will then be provided to the tool according to the predefined fields. The aim is to create an ABAC authoring environment of business-level user experience while delegating the transformation of the provided input to machine-readable ABAC rules to the application. Unlike our proposal, following Turner's approach, information relevant to ABAC attributes is still extracted manually. Although they mentioned that this task could be automated, their approach is manual. Additionally, Turner's approach constrains the NLACP's context-free grammar while we consider unrestricted natural language policies.

In comparison with the above-mentioned studies, our work is the first to evaluate the automation task in the context of ABAC.

ABAC policy mining

Policy mining algorithms were proposed to automate (or semi-automate) the migration from traditional access control mechanisms to a target access control model, be it ABAC or role-based access control (RBAC). In the context of ABAC, the mining task is mostly a two-phase process: (1) generating the candidate rules and (2) consolidating the resulting rules to optimize policy goodness measures. In this field of research, studies could be grouped into three main themes: heuristic-based, machine learning-based and evolutionary-based algorithms. Xu et al. proposed several heuristic-based policy mining algorithms to construct ABAC rules given various forms of access control models (Xu and Stoller 2013; 2014; 2015). Later, Gautam et al. proposed to model the ABAC mining problem as Constrained Weighted Set Cover Heuristic to better optimize the mined policy (Gautam et al. 2017). While Xu and Gautam focused on mining affirmative ABAC authorization rules, Padmavathi et al. proposed a more systematic, yet heuristic, approach to mine both positive and negative rules (Iyer and Masoumzadeh 2018). Mocanu et al. first designed and evaluated an ABAC candidate rules generation framework using machine learning techniques. This approach benefits from the capabilities of the generative learning algorithms to generate samples of candidate rules, given information of user and resource attributes as well as the input policy (e.g. access control list) (Mocanu et al. 2015). Benkaouz et. al. used KNN to cluster the candidate rules into groups of similar rules. Then, the representative rules of each cluster are selected to form the mined ABAC system (Benkaouz et al. 2016). ABAC mining problem has also been addressed using an evolutionary approach with which each individual in the population is a candidate rule (Medvet et al. 2015). To grow the set of initial rules, the authors defined mutation and crossover operations to be operated over the initial set. The resulted rules are associated with fitness value that quantifies the goodness of the rules.

With this, it is clear that ABAC policy mining is similar to the spirit of our work as both targets the automated development of ABAC rules. The primary downside of applying mining solutions is its significant lack of semantics. Further, mining algorithms often require prior knowledge of the target model, e.g., the attributes of subjects and objects, to define the set of rules. Contrary to that, our work follows a top-down engineering approach which is preferred for establishing a more representative model. It is also worthy to mention that our approach does not assume any external knowledge regarding the policy. Instead, it initiates the process directly on natural language requirement specifications.

Conclusions and future work

A practical framework was proposed to analyze natural language ACPs, and automatically identify the core properties needed to define the attributes contained in those policies. To the best of our knowledge, we present the first evaluated effort to solve this problem. Our proposed approach is built upon natural language processing and machine learning techniques and in addition to identifying attributes, maps attribute values to attribute keys. When extracting the values of subject and object attributes, our approach achieved an average F1-score of 0.96 and 0.91, respectively. We also obtained an average F1-score of 0.85 and 0.81 in the task of constructing the value space of subject attributes and object attributes, respectively. The relatively high values of the F1-measure indicate that our proposed method is promising. While the current results are encouraging, further data collection effort is needed to improve the performance over out-of-domain data. Our future research directions are focused on devising automated means in order to better support the overall ABAC policy authoring experience.

Endnotes

[1] Action attribute has been added for the sake of completeness of our example though it was not particularly part of this work. Extending the current framework to include action element is feasible in the presence of required data

[2] Hypernym is an expression of a more generic concept compared with a given word (see http://www.nltk.org/howto/wordnet_lch.html)

[3] These documents can be downloaded from: https://sites.google.com/site/accesscontrolruleextraction/documents

Abbreviations

ABAC: Attribute-based access control; AC: Access control; ACP: Access control policy; ACRE: Access control relation extraction; AP: Affinity propagation; APIs: Application programming interface; Aucc: Accuracy; AV: Attribute value; CNN: Convolutional neural network; DBSAN: Density based spatial clustering of application with noise; E: Element; Eps: Epsilon; FN: False negatives; FP: False positive; GUI: Graphical user interface; KNN: K-nearest neighbors; Minpts: Minimum number of points; ML: Machine learning; N/A: Not applicable; N/R:

Not reported; NB: Naive bayes; NE: Named entity; NER: Named entity recognition; NIST: National institute of standards and technology; NLACP(s): Natural language access control policy(ies); NLP: Natural language processing; OASIS: Organization for the advancement of structured information standards; P or Prec: Precision; POS: Part of speech; R or Rec: Recall; RE: Relation extraction; RQ: Research question; SRL: Semantic role labeler; SVM: Support vector machine; TP: True positive; XACML: eXtensible access control markup language

Acknowledgements
Not applicable.

Funding
The work of the first author has been supported by Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia.

Availability of data and materials
The datasets supporting the conclusions of this article are available upon request. Please email the first author for this matter.

Authors' contributions
All the authors contributed equally. All authors read and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Received: 5 September 2018 Accepted: 18 December 2018
Published online: 21 January 2019

References

Abassi R, Rusinowitch M, Jacquemard F, El Fatmi SG (2010) Xml access control: from xacml to annotated schemas. In: The Second International Conference on Communications and Networking. IEEE, Tozeur. pp 1–8. https://ieeexplore.ieee.org/document/5699810

Alohaly M, Takabi H, Blanco E (2018) A Deep Learning Approach for Extracting Attributes of ABAC Policies. In: Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies, ACM, New York, IN, USA, SACMAT '18. pp 137–148. https://dl.acm.org/citation.cfm?doid=3205977.3205984

Axiomatics (2017) Attribute based access control (ABAC). https://www.axiomatics.com/attribute-based-access-control/. Accessed 2018

Bakhshandeh O, Allen J (2015) From adjective glosses to attribute concepts: Learning different aspects that an adjective can describe. In: Proceedings of the 11th International Conference on Computational Semantics, Association for Computational Linguistics, London, UK IWCS. pp 23–33. http://aclweb.org/anthology/W15-0103

Banea C, Chen D, Mihalcea R, Cardie C, Wiebe J (2014) Simcompass: Using deep learning word embeddings to assess cross-level similarity. In: Proceedings of the 8th International Workshop on Semantic Evaluation, Association for Computational Linguistics,Dublin, Ireland, (SemEval 2014). pp 560–565. http://aclweb.org/anthology/S14-2098

Bengio Y, Ducharme R, Vincent P, Jauvin C (2003) A neural probabilistic language model. J Mach Learn Res 3(Feb):1137–1155

Benkaouz Y, Erradi M, Freisleben B (2016) Work in progress: K-nearest neighbors techniques for ABAC policies clustering. In: Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control, ACM, New York, NY, USA, ABAC '16. pp 72–75. https://doi.org/10.1145/2875491.2875497, http://doi.acm.org/10.1145/2875491.2875497

Berland M, Charniak E (1999) Finding parts in very large corpora. In: Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics on Computational Linguistics, Association for Computational Linguistics, Stroudsburg, PA, USA, ACL '99. pp 57–64. https://doi.org/10.3115/1034678.1034697

Brill E (1995) Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. Comput Linguist 21(4):543–565. http://dl.acm.org/citation.cfm?id=218355.218367

Brossard D, Gebel G, Berg M (2017) A systematic approach to implementing ABAC. In: Proceedings of the 2Nd ACM Workshop on Attribute-Based Access Control, ACM, New York, NY, USA, ABAC '17. pp 53–59. https://doi.org/10.1145/3041048.3041051, http://doi.acm.org/10.1145/3041048.3041051

Chen D, Manning C (2014) A fast and accurate dependency parser using neural networks. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). Association for Computational Linguistics, Doha. pp 740–750. http://aclweb.org/anthology/D14-1082

Collobert R, Weston J (2008) A unified architecture for natural language processing: Deep neural networks with multitask learning. In: Proceedings of the 25th International Conference on Machine Learning, ACM, New York, NY, USA, ICML '08. pp 160–167. https://doi.org/10.1145/1390156.1390177, http://doi.acm.org/10.1145/1390156.1390177

Collobert R, Weston J, Bottou L, Karlen M, Kavukcuoglu K, Kuksa P (2011) Natural language processing (almost) from scratch. J Mach Learn Res 12(Aug):2493–2537

Common crawl (2018). http://commoncrawl.org/. Accessed 2018

Culotta A, Sorensen J (2004) Dependency tree kernels for relation extraction. In: Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics, Association for Computational Linguistics, Stroudsburg, PA, USA, ACL '04. https://doi.org/10.3115/1218955.1219009

Dandelion api (2018). https://dandelion.eu/. Accessed 2018

De Marneffe MC, Manning CD (2008) Stanford typed dependencies manual. Tech. Stanford University, Technical report

Dingwall N, Potts C (2018) Mittens: An extension of glove for learning domain-specialized representations. arXiv preprint arXiv:180309901

Ester M, Kriegel HP, Sander J, Xu X, et al. (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: Kdd, vol 96. ACM, Portland. pp 226–231. https://dl.acm.org/citation.cfm?id=3001507

Frey BJ, Dueck D (2007) Clustering by passing messages between data points. Science 315(5814):972–976. https://doi.org/10.1126/science.1136800, http://science.sciencemag.org/content/315/5814/972

Gartner (2013) Market trends: Cloud-based security services market, worldwide, 2014. https://www.gartner.com/doc/2607617. Accessed 2018

Gautam M, Jha S, Sural S, Vaidya J, Atluri V (2017) Poster: Constrained policy mining in attribute based access control. In: Proceedings of the 22Nd ACM on Symposium on Access Control Models and Technologies, ACM, New York, NY, USA, SACMAT '17 Abstracts. pp 121–123. https://doi.org/10.1145/3078861.3084163, http://doi.acm.org/10.1145/3078861.3084163

Glasser J, Lindauer B (2013) Bridging the gap: A pragmatic approach to generating insider threat data. In: 2013 IEEE Security and Privacy Workshops. IEEE, San Francisco. pp 98–104. https://doi.org/10.1109/SPW.2013.37

Guo Q, Vaidya J, Atluri V (2008) The role hierarchy mining problem: Discovery of optimal role hierarchies. In: 2008 Annual Computer Security Applications Conference (ACSAC). IEEE, Anaheim. pp 237–246. https://doi.org/10.1109/ACSAC.2008.38

GuoDong Z, Jian S, Jie Z, Min Z (2005) Exploring various knowledge in relation extraction:427–434. https://doi.org/10.3115/1219840.1219893

Hartigan JA, Wong MA (1979) Algorithm as 136: A k-means clustering algorithm. J R Stat Soc Ser C (Appl Stat) 28(1):100–108

Hu VC, Ferraiolo D, Kuhn R, Friedman AR, Lang AJ, Cogdell MM, Schnitzer A, Sandlin K, Miller R, Scarfone K, et al. (2013) Guide to attribute based access control (ABAC) definition and considerations. NIST Spec Publ 800(162)

IBM (2004) Course registration requirements. https://khanhn.files.wordpress.com/2016/08/vidu-ibm.pdf. Accessed 2018

Iyer P, Masoumzadeh A (2018) Mining positive and negative attribute-based access control policy rules. In: Proceedings of the 23Nd ACM on Symposium on Access Control Models and Technologies, ACM, New York, NY, USA, SACMAT '18. pp 161–172. https://doi.org/10.1145/3205977.3205988, http://doi.acm.org/10.1145/3205977.3205988

Jiang J (2012) Information extraction from text. In: Aggarwal CC, Zhai C (eds). Data, Mining Text. Springer, U S, Boston, MA. pp 11–41

Johansson R, Nugues P (2008) Dependency-based semantic role labeling of propbank. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Stroudsburg, PA, USA, EMNLP '08,. pp 69–78. http://dl.acm.org/citation.cfm?id=1613715.1613726

Jurafsky D, Martin JH (2009) Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition. Prentice Hall series in artificial intelligence. pp 1–1024

Kalchbrenner N, Blunsom P (2013) Recurrent convolutional neural networks for discourse compositionality. CoRR abs/1306:3584. http://arxiv.org/abs/1306.3584, 1306.3584

Karol S, Mangat V (2013) Evaluation of text document clustering approach based on particle swarm optimization. Open Comput Sci 3(2):69–90

Lampson BW (1974) Protection. SIGOPS Oper Syst Rev 8(1):18–24. https://doi.org/10.1145/775265.775268, http://doi.acm.org/10.1145/775265.775268

LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436

Martin JH, Jurafsky D (2000) Speech and Language processing: An introduction to natural language processing. Computational Linguistics and Speech Recognition, Prentice Hall, 2

McCarthy V (2003) Xacml a no-nonsense developer's guide. http://www.idevnews.com/stories/57. Accessed 2018

Medvet E, Bartoli A, Carminati B, Ferrari E (2015) Evolutionary inference of attribute-based access control policies. In: International Conference on Evolutionary Multi-Criterion Optimization, Springer International Publishing, Cham, EMO (1). pp 351–365

Meneely A, Williams L, Smith B (2011) itrust electronic health care system: A case study. http://bensmith.s3.amazonaws.com/website/papers/sst2011.pdf. Accessed 2018

Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient Estimation of Word Representations in Vector Space. In: Proceedings of ICLR Workshops Track. ArXiv e-prints, arxiv.org/abs/1301.3781

Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013) Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems. Curran Associates, Inc. pp 3111–3119

Miller GA (1995) Wordnet: a lexical database for english. Commun ACM 38(11):39–41

Mnih A, Hinton GE (2009) A scalable hierarchical distributed language model. In: Koller D, Schuurmans D, Bengio Y, Bottou L (eds). Advances in Neural Information Processing Systems 21,. Curran Associates, Inc. pp 1081–1088. http://papers.nips.cc/paper/3583-a-scalable-hierarchical-distributed-language-model.pdf

Mocanu D, Turkmen F, Liotta A (2015) Towards ABAC policy mining from logs with deep learning. In: Proceedings of the 18th International Multiconference, IS 2015, Intelligent Systems, Ljubljana

Narouei M, Takabi H (2015) Automatic top-down role engineering framework using natural language processing techniques. In: IFIP International Conference on Information Security Theory and Practice. Springer International Publishing, Cham. pp 137–152. https://link.springer.com/chapter/10.1007/978-3-319-24018-3_9

Narouei M, Takabi H (2015a) Towards an automatic top-down role engineering approach using natural language processing techniques. In: Proceedings of the 20th ACM Symposium on Access Control Models and Technologies, ACM, New York, NY, USA, SACMAT '15. pp 157–160. https://doi.org/10.1145/2752952.2752958, http://doi.acm.org/10.1145/2752952.2752958

Narouei M, Khanpour H, Takabi H (2017) Identification of access control policy sentences from natural language policy documents. In: Livraga G, Zhu S (eds). Data and Applications Security and Privacy XXXI. Springer International Publishing, Cham, DBSec. pp 82–100

Narouei M, Khanpour H, Takabi H, Parde N, Nielsen R (2017) Towards a top-down policy engineering framework for attribute-based access control. In: Proceedings of the 22Nd ACM on Symposium on Access Control Models and Technologies, ACM, New York, NY, USA, SACMAT '17. pp 103–114. https://doi.org/10.1145/3078861.3078874. http://doi.acm.org/10.1145/3078861.3078874

Narouei M, Takabi H, Nielsen R (2018) Automatic extraction of access control policies from natural language documents. IEEE Transactions on Dependable and Secure Computing. IEEE. pp 1–1. https://doi.org/10.1109/TDSC.2018.2818708

Oak M, Behera A, Thomas T, Alm CO, Prud'hommeaux E, Homan C, Ptucha RW (2016) Generating clinically relevant texts: A case study on life-changing events. In: Proceedings of the Third Workshop on Computational Lingusitics and Clinical Psychology. pp 85–94

OASIS (2013) extensible access control markup language (xacml) version 3.0. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

Pelletier FJ (1994) The principle of semantic compositionality. Topoi 13(1):11–24. https://doi.org/10.1007/BF00763644

Pennington J, Manning CD, Socher R (2017) Glove: Global vectors for word representation. https://nlp.stanford.edu/projects/glove/

Pennington J, Socher R, Manning C (2014) Glove: Global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). pp 1532–1543. http://www.aclweb.org/anthology/D14-1162

Rehman SU, Asghar S, Fong S, Sarasvady S (2014) Dbscan: Past, present and future. In: The Fifth International Conference on the Applications of Digital Information and Web Technologies (ICADIWT 2014). IEEE, Bangalore. pp 232–238. https://doi.org/10.1109/ICADIWT.2014.6814687

Schubert E, Sander J, Ester M, Kriegel HP, Xu X (2017) Dbscan revisited, revisited: Why and how you should (still) use dbscan. ACM Trans Database Syst 42(3):19:1–19:21

Servos D, Osborn SL (2015) Hgabac: Towards a formal model of hierarchical attribute-based access control. In: Cuppens F, Garcia-Alfaro J, Zincir Heywood N, Fong PWL (eds). Foundations and Practice of Security, Springer International Publishing, Cham. pp 187–204. https://link.springer.com/chapter/10.1007/978-3-319-17040-4_12#citeas

Shen Y, He X, Gao J, Deng L, Mesnil G (2014) Learning semantic representations using convolutional neural networks for web search. In: Proceedings of the 23rd International Conference on World Wide Web, ACM, New York, NY, USA, WWW '14 Companion. pp 373–374. https://doi.org/10.1145/2567948.2577348, http://doi.acm.org/10.1145/2567948.2577348

Slankas J, Williams L (2013) Access control policy identification and extraction from project documentation. SCIENCE 2(3):145–159

Slankas J, Xiao X, Williams L, Xie T (2014) Relation extraction for inferring access control rules from natural language artifacts. In: Proceedings of the 30th Annual Computer Security Applications Conference, ACM, New York, NY, USA, ACSAC '14. pp 366–375. https://doi.org/10.1145/2664243.2664280, http://doi.acm.org/10.1145/2664243.2664280

Sokolova M, Lapalme G (2009) A systematic analysis of performance measures for classification tasks. Inf Process Manag 45(4):427–437. https://doi.org/10.1016/j.ipm.2009.03.002, http://www.sciencedirect.com/science/article/pii/S0306457309000259

Steinbach M, Karypis G, Kumar V, et al. (2000) A comparison of document clustering techniques. In: KDD workshop on text mining, Boston, vol 400. pp 525–526. http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.125.9225

Stoller SD, Bui T (2016) Mining hierarchical temporal roles with multiple metrics. In: Proceedings of the 30th Annual IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy. Springer International Publishing, Cham, DBSec16. pp 79–95

Tjong Kim Sang EF, De Meulder F (2003) Introduction to the conll-2003 shared task: Language-independent named entity recognition. In: Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4, Association for Computational Linguistics, Stroudsburg, PA, USA, CONLL '03. pp 142–147. https://doi.org/10.3115/1119176.1119195

Tran TN, Drab K, Daszykowski M (2013) Revised dbscan algorithm to cluster data with dense adjacent clusters. Chemometr Intell Lab Syst 120:92–96

Turner RC (2017) Proposed model for natural language ABAC authoring. In: Proceedings of the 2Nd ACM Workshop on Attribute-Based Access Control, ACM, New York, NY, USA, ABAC '17. pp 61–72. https://doi.org/10.1145/3041048.3041054, http://doi.acm.org/10.1145/3041048.3041054

Van De Stadt R (2012) Cyberchair: A web-based groupware application to facilitate the paper reviewing process. CoRR abs/1206.1833. http://arxiv.org/abs/1206.1833, withdrawn., 1206.1833

Webster JJ, Kit C (1992) Tokenization as the initial phase in nlp. In: Proceedings of the 14th Conference on Computational Linguistics - Volume 4, Association for Computational Linguistics, Stroudsburg, PA, USA, COLING '92. pp 1106–1110. https://doi.org/10.3115/992424.992434

Xiao X, Paradkar A, Thummalapenta S, Xie T (2012) Automated extraction of security policies from natural-language software documents. In: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, ACM, New York, NY, USA, FSE '12. pp 12:1–12:11. https://doi.org/10.1145/2393596.2393608, http://doi.acm.org/10.1145/2393596.2393608

Xu Z, Stoller SD (2013) Mining attribute-based access control policies from rbac policies. In: Proceedings of the 10th International Conference and Expo on Emerging Technologies for a Smarter World CEWIT 2013. IEEE. pp 1–6

Xu Z, Stoller SD (2014) Mining attribute-based access control policies from logs. In: IFIP Annual Conference on Data and Applications Security and Privacy, Springer. pp 276–291. https://link.springer.com/chapter/10.1007/978-3-662-43936-4_18

Xu Z, Stoller SD (2015) Mining attribute-based access control policies. IEEE Trans Dependable Secure Computing 12(5):533–545. https://ieeexplore.ieee.org/document/6951368

Zelenko D, Aone C, Richardella A (2003) Kernel methods for relation extraction. J Mach Learn Res 3(Feb):1083–1106
Zeng D, Liu K, Lai S, Zhou G, Zhao J, et al. (2014) Relation classification via convolutional deep neural network. In: 2014, the 25th International Conference on Computational Linguistics. Dublin City University and Association for Computational Linguistics, Dublin. pp 2335–2344
Zhang M, Zhang J, Su J (2006) Exploring syntactic features for relation extraction using a convolution tree kernel. In: Proceedings of the Main Conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, Association for Computational Linguistics, Stroudsburg, PA, USA, HLT-NAACL '06. pp 288–295. https://doi.org/10.3115/1220835.1220872