Access Control Policy Identification and Extraction from Project Documentation

John Slankas and Laurie Williams
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA
Email: [john.slankas,laurie_williams]@ncsu.edu

ABSTRACT

While access control mechanisms have existed in computer systems since the 1960s, modern system developers often fail to ensure appropriate mechanisms are implemented within particular systems. Such failures allow for individuals, both benign and malicious, to view and manipulate information that they should not otherwise be able to access. *The goal of our research is to help developers improve security by extracting the access control policies implicitly and explicitly defined in natural language project artifacts.* Developers can then verify and implement the extracted access control policies within a system. We propose a machine-learning based process to parse existing, unaltered natural language documents, such as requirement or technical specifications to extract the relevant subjects, actions, and resources for an access control policy. To evaluate our approach, we analyzed a public requirements specification. We had a precision of 0.87 with a recall of 0.91 in classifying sentences as access control or not. Through a bootstrapping process utilizing dependency graphs, we correctly identified the subjects, actions, and objects elements of the access control policies with a precision of 0.46 and a recall of 0.54.

I Introduction

Despite significant remediation efforts over the past decade, such as those due to information technology controls required for Sarbanes-Oxley [1], and the highlighting of access control errors in lists such lists as the CWE/SANS Top 25 Most Dangerous Software Errors [2], access control remains a significant issue. In the 2013 Verizon Data Breach Investigations Report [3], 61% of the incidents included some form of access control abuse. Just as significant, 60% of the breaches went undiscovered for months. Edward Snowden's public disclosure of the National Security Agency's PRISM program [4] demonstrates the challenges organizations face in keeping information secure. In this situation, Snowden inappropriately accessed electronic documents he should not have been able to access.

To mitigate data security issues, organizations must properly implement access control across all system components. Access control is a critical application mechanism to ensure confidentiality and integrity [5]. While various access control models exist (discretionary, mandatory, role-based, attribute, etc.), most models contain a tuple (subject, resource, action) to represent a policy as to whether or not the subject (a user) can perform the requested action on the specified resource (object) within the system. Access control policies are often expressed, implicitly or explicitly, within nature language. For example, "The system shall allow hospital administrators to inactive patients" explicitly grants users who are hospital administrators the ability to inactivate patients. However, creating and defining the correct access control policies can be a tedious, time-consuming, and error-prone endeavor. Developers must extract access control policies from existing documentation, application code, and database implementations. As an extreme example, the United States Department of Defense has over 180 different policy documents from 16 sources for its Trusted Global Information Grid [6].

The goal of our research is to help developers improve security by extracting the access control policies implicitly and explicitly defined in natural language project artifacts.

We propose a process, which we call Access Control Relation Extraction (ACRE), which allows organizations to utilize existing, unconstrained natural language text to extract

access control policies. ACRE analyzes requirements or other natural language statements to obtain their subject, action, and resource elements. For each sentence, ACRE first parses the sentence into a dependency graph that represents the syntactic relationships between words within the sentence. Next, the process makes a decision about whether the sentence is related to access control or not. If the sentence is considered to involve access control, then the process utilizes relations among the words to extract subjects, actions, and resources based upon initial and learned patterns within the dependency graph. From the actions and other information in the pattern, the permissions can be inferred for the access control policy. Once the appropriate natural language documents have been examined, the process validates the extracted access control policies.

The process utilizes a combination of natural language processing (NLP), information extraction (IE), and machine learning (ML) techniques. A critical component to our process is the generation of appropriate dependency graph patterns based upon an initial set of seeded patterns and then expanding the set of patterns through extracting new patterns where combinations of any discovered access control elements (subjects, actions, resources) can be located in the current document under investigation. This approach of learning new patterns from an initial set of seed patterns is termed "bootstrapping" [7]. Due to the ambiguity and multitude of different ways of representing concepts within natural language, our process allows for humans to enter undiscovered patterns or correct patterns misidentified in the bootstrapping approach.

To evaluate our process, we developed the following research questions:

RQ1: How effectively can we identify access control policies in natural language text in terms of precision and recall?
RQ2: What common patterns exist in sentences expressing access control policies?
RQ3: What is an appropriate set of seeded graphs to effectively bootstrap the process to extract the access control elements?

We evaluated our process and tool against an open source educational testbed, the iTrust Electronic Health Records System[1] [8].

Our research contributes the following:

- A bootstrapping approach for to seed and discover access control patterns in natural language text.
- A minimal document grammar to produce additional contextual information for titles and lists in documents.
- Process and tool to extract access control elements in natural language text.

The rest of this paper is organized as follows: Section II provides requisite background information. Section III presents related work. Next, Section IV details ACRE and the associated tool. Section V presents our methodology. Section VI presents our evaluation of the approach. Section VII discusses limitations. In Section VIII, we discuss future work. Finally, we conclude in Section IX.

II Background

This section presents on overview on the material necessary to understand paper's approach to extracting access control policy and evaluating the effectiveness.

1 Machine Learning and Classification

To identify access control related statements from unconstrained natural language texts, we need flexible, yet effective classification methods to handle different documents and multiple ways of expressing similar concepts. Machine learning provides such a foundation for our work. While techniques and algorithms vary widely in machine learning, they can be generally divided into two primary categories: supervised learning and unsupervised learning. In supervised learning, people train classifiers with labeled data. People and systems then use these classifiers to decide in which class a previously unseen instance belongs. In contrast, unsupervised learning algorithms search data for common patterns (clusters). The

[1] http://agile.csc.ncsu.edu/iTrust/

data is not directly labeled, but rather groups of common instances are created.

For this work, we utilized a k-nearest neighbor classifier (k-NN), which is a supervised learning algorithm. k-NN classifiers work by classifying a test item based upon which items previously classified are closest to the current test item. The classifier finds the k nearest "neighbors" and returns a majority vote of those neighbors to classify the test item. A distance metric determines the closeness between two items. Euclidean distance often serves as a metric for numerical attributes. For nominal values, the distance is binary – zero if the values are the same or one if the values differ. k-NN classifiers may use custom distance functions specific to the current problem. Advantages of k-NN classifiers include the ability to incrementally learn as new items are classified, to classify multiple types of data, and to handle large numbers of item attributes. The primary drawback of k -NN classifiers involves algorithm runtime; if the classifiers have n items stored, classification takes $O(n)$ time.

We evaluated other supervised learning algorithms including naïve Bayes, Support Vector Machines (SVM), and term frequency – inverse document frequency (TF-IDF). A naïve Bayes classifier works by selecting a class with the highest probability from a set of trained data sets given a specific document. Fundamentally, it assumes that each feature of a class exists independently of other features. Despite the simplification, the approach performs effectively in real-world problems. Naïve Bayes classifiers typically require fewer trained instances than other classifiers. SVM classifiers work by finding the optimal separator between two classes. As with naïve Bayes, text is represented as a word vector [9]. Within TF-IDF, each word belonging to a category is represented by two values: (1) the frequency a given term appears within a given document; and (2) the frequency of the total number of documents over the number of documents containing a specific term. The second value places a higher premium on terms appearing in fewer documents.

2 Classification Evaluation

To compare the results, we used recall, precision, and the F_1 measure. To compute these values, we first need to categorize the classifier's predictions into three categories for each classification value. True positives (TP) are correct predictions. False positives (FP) are predictions in which the sentence of another classification is classified as the one under evaluation. False negatives (FN) are predictions in which a sentence of the same classification under evaluation is placed into another classification. From these values, we define precision (P) as the proportion of corrected predicted classifications against all predictions against the classification under test. We define recall as the proportion of classifications found for the current classification under test. The F_1 measure is the harmonic mean of precision and recall, giving an equal weight to both elements. The specific formulas are defined in Fig. 1. From an access control perspective, high values for both precision and recall are desired. Lower precision implies that the process will ultimately grant a role more incorrect permissions than a classification with higher precision. Lower recall implies that we will have missed access control sentences.

$$P = \frac{TP}{TP + FP}$$

$$R = \frac{TP}{TP + FN}$$

$$F_1 = 2 \times \frac{P \times R}{P + R}$$

Figure 1: Formulas for Precision, Recall, and F_1

3 Information Extraction

IE is an area of NLP concerned with finding structured data from text [7]. IE differs from information retrieval (e.g. web searches) in that IE's goal is to extract information rather than provide a ranked list of relevant documents to a query. Common IE tasks include named entity recognition, reference resolution, relation extraction (RE) and event extraction. A relation expresses the relationship between two entities. Common relation types include "is-a" and "part-of". For example, "a bicycle is a vehicle" is represented by *is_a(bicycle, vehicle)* and bicycles have two wheels is represented by *contains(bicycle, wheels)*. Similarly, we can have relation *writes(doctor, prescription)* to indicate that a doctor can write a prescription. The IE field has advanced largely due to the investments by the United States government through

challenges sponsored by DARPA [10] and NIST[2]. State of the art systems for RE in these challenges typically have around 85% precision and 70% recall [11].

Access control policy extraction is most similar to the RE task in that the relation (action) between the subject and resource implies much of the access control policy within a statement. However, access control extraction differs from most RE task in that it is not constrained by small, fixed sets of binary relations [7]. Additionally, access control extraction needs to infer the appropriate permissions based upon the identified relation (action). The permission may be further constrained by other features within the sentences such as negativity or access limitations to just a particular person.

III Related Work

This section reviews the work related to our research.

1 Natural Language and Access Control

Other researchers have explored using natural language to generate access control policies from natural language. He and Antón [12] proposed an approach based upon available project documents, database design, and existing policies. Utilizing a series of heuristics, humans would analyze the documents to find additional access control policies. In addition to heuristics to find the elements within the typical access control tuple (subject, resource, action), they created heuristics to identify policy constraints (temporal, location, relationship, privacy, etc.) and obligations. More recently, Xiao et al. [13] present an approach, Text2Policy, where they parsed use cases to create eXtensible Access Control Markup Language[3] (XACML) policies. Their approach was specific to use case-based requirement specifications and relied upon matching four specific sentence patterns to deduce the necessary information to populate an access control method.

2 Controlled Natural Language

Other researchers have resolved converting natural language to and from policies by utilizing a controlled natural language (CNL). Schwitter [14] defines CNLs as "engineered subsets of natural languages whose grammar and vocabulary have been restricted in a systematic way in order to reduce both ambiguity and complexity of full natural languages." While CNLs provide consistent, semantic interpretations, CNLs limit authors and typically require language specific tools to stay within the constraints of the language. Project documents previously created cannot be used as inputs without processing the documents manually into the tools. Policies authored outside of tools must confirm to strict limited grammars to be automatically parsed as well. Brodie et al. [15] used this approach in the SPARCLE Policy Workbench. By using their own natural language parser and a controlled grammar, they were effectively able to translate from controlled natural language into formal policy. Users responded favorably to their policy authoring tool. Inglesant et al. [16] demonstrated similar success with their tool, PERMIS, which utilizes a RBAC authorization model. However, they did report issues with users not understanding the predefined "building blocks" imposed by using a CNL. Recently, Shi and Chadwick [17] presented their results of an application to author access control policies using a CNL. While they showed the improved usability of CNL interface, they were limited in the complexity of the policies that could be created as the interface did not support conditions or obligations. Our approach utilizes project artifacts without any change to the vocabulary or structure of the sentences.

3 Relation Extraction

A number of different ways exist to identify semantic relations within text. Initial solutions employed hand-written patterns to detect hyponym ("is-a") relationships among words [18]. While hand-written patterns usually have high precision, they tend to suffer low recall by missing relation occurrences. Snow et al. [19] used dependency paths and grammatical relationships within a sentence to discover additional relationship patterns. Akbik and Broß [20] used a similar process to extract a diverse range of sematic relations with the goal to extract

[2] http://www.itl.nist.gov/iad/mig/tests/ace/

[3] http://www.oasis-open.org/committees/xacml

arbitrary relations for semantic search. Zhou et al. [21] documented a variety of possible features to use in relation classification. Dependency parse tree patterns have been used to extracting relations between genes and proteins [22]. Recent research with regards to information extraction has focused on gathering a wide range of relations from extremely large document collections (e.g., portions of the Internet such as Wikipedia). Such approaches typically use distance supervision (i.e., a trusted source of known relations) to evaluate discovered patterns [23]. Wu and Weld [24] extended this approach and utilized Stanford's NLP to extract the shortest dependency path between two words. They then trained a classifier to determine if the extracted relations are valid. More recently, Mausam et al. [25] developed a new system, OLLIE that allows for relations to mediated by parts of speech other than nouns and provides support for contextual modifiers to determine if relations are correct. Our work builds primarily upon the last work. We utilize dependency graphs, but extract the minimum graph pattern that overlays the lowest common ancestor of all of the included elements of the dependency graph. These elements include the subject, action, the relation, any indication of negativity, any indication of access limited to a particular subject, and any contextual information necessary to disambiguate the necessary permissions for the access control policy. As with other works [19], we include capability to bootstrap the possible graph patterns and a naïve Bayes classifier to judge whether such derived patterns should be include in the evaluation set.

IV Access Control Relation Extraction

This section details our process, Access Control Relation Extraction (ACRE), to extract access control policies from natural language text. We first present how access control appears within natural language text, then our representation of access control policy, and then the overall process.

1 Access Control and Natural Language

Within natural language texts, access control elements are both explicitly and implicitly stated. For example, "The system shall allow patients to view their own medical records" explicitly grants users who are patients the right to read their medical records. Other sentences such as "A nurse can order a lab procedure for a patient" implies two access control policies. First, the nurse has some form of create or write permission for lab procedures. Secondly, since the lab procedure is for a patient, another access control policy implicitly exists to grant the nurse read access to patients. In many situations, the verb that typically represents the action within the sentence implies the necessary permissions to be granted. However, in some cases, the permissions are not necessarily so straightforward. As with the second example, the verb "order" combined with the prepositional phrase, "for the patient", implies read access to a patient. Another example of the need to utilize other contextual information is the sentence, "The doctor may add or remove patients from the monitoring list." Naively, one could assume that doctors have the right to add or remove patients. In this sentence, though, the doctor is granted the right to manipulate the monitoring list by adding or removing patients. Additionally, we assume that the doctor has a read permission on a patient object in order to perform the former activities.

Denying users permission to an object when they should not have access to that object is a critical feature of any access control system. As such, we need to develop patterns to detect when permission should be explicitly revoked from a particular subject in an access control policy. From the Dixon's negation concepts [26], we utilize multiple methods to detect negation within a sentence. We can detect specific adjectives (unable), adverbs (not, never), determiners (no, zero, neither), and nouns (none, nothing) [27]. We also utilize specific negative verbs (stop, prohibit, forbid) as well as a pre-determined list of negative prefixes associated with English words.

Within a system, privileges are often limited to one or more specific roles. For the sentence "only an administrator can maintain system lookup tables", we need to grant permission to the administrator role while restricting the permission from all other roles. Within English, restrictive focus modifiers [27] express such limitations. As the position of such modifiers (e.g. just, only) affect the overall semantic meaning of the sentence, we restrict where such modifiers may be placed. For example, consider the sentence, "Doctors write prescriptions." We assume the doctor has the privilege to write prescriptions

(possibly an insert permission on a "prescription" table within a relational database). However, if the word "only" is placed into the sentence, then the meaning of the sentence varies based upon the location of "only". With only as the first word, the sentence means that only doctors write prescriptions and no other roles. Alternatively, if only is the second word, it modifies "write" and implies that the only action doctors can do in the system is write prescriptions. Finally, if only is the third word, the sentence now means that prescriptions are the only things doctors write. To ensure the modifiers are in the correct location, we set the limit flag for a sentence if the modifier applies to the subject identified in the sentence. If the modifier exists elsewhere, we flag the sentence for having a potential problem.

2 ACRE Access Control Policy Representation

Internally, we represent sentences with a dependency graph as depicted in Fig. 2 for the sentence "a nurse can order a lab procedure for a patient." (In the next section, we discuss how these graphs are produced.) Each vertex represents a word from the sentence along with the word's part of speech. In the figure, "NN" represents a noun, "VB" represents a verb, and "MD" represents a modal verb. Edges represent the grammatical relationship between two words. For instance, "nurse" functions as the nominal subject (nsubj) for "order" and "lab procedure" is the direct object (dobj) to be ordered. Dependency graphs can be considered trees in most situations and are typically rooted by the sentence's main verb. When conjunctions are present, nodes may have multiple parents and, thus, need to be treated as graphs.

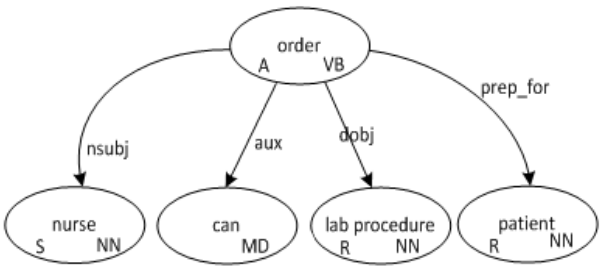

Figure 2: ACRE Sentence Representation

To represent an access control policy, we utilize the following pattern, termed an "access control pattern":

$$A(\{s\},\{a\},\{r\},[n],[l],\{c\},H,p)$$

Figure 3: Access Control Representation

A defines the overall access control policy. s contains an order set of vertices that compose the subject of a policy. Similarly, a and r represent the action and resource, respectively. n contains the vertex representing negativity if required for the policy. If the policy should be limited to a particular subject s, l contains the indicating vertex. c contains any additional vertices required to provide context to given action for a set of permissions. H represents the subgraph of a sentence's dependency graph that contains the vertices and necessary edges to connect all of the vertices listed in s, a, r, n, l, c. p represents the permissions typically associated with an action. We limit permissions to have the values of "create", "retrieve", "update", and "delete" as we are primarily concerned with controlling the ability to view and manipulate data in systems. From the example in Fig. 2, we define these two policies:

$A((nurse),(order),(lab\ procedure),(\,),(\,),(V: nurse, order, lab\ procedure;$
$E: (order, nurse, nsubj); (order, lab\ procedure, dobj)\,), create)$
$A((nurse),(order),(patient),(\,),(\,),(V: nurse, order, patient;$
$E: (order, nurse, nsubj); (order, patient, prep\_for)\,), read)$

Figure 4: Extracted Access Control Policies

Situations exist in which not all of the access control policy elements may be present within a single sentence. These access control policies may be identified with missing elements. Developers would be pointed to the surrounding sentences to finish defining the access control policy.

3 Access Control Relation Extraction Process

The ACRE process consists of five primary steps:

1. Parse text document
2. Parse natural language
3. Classify sentence as access control or not
4. Extract access control elements
5. Validate access control policies

Step 1: Parse Text Document

The process first reads the entire text into the system. We then separate the input into tokens by either new lines or by periods at the end of sentences. Next, we apply a concise document grammar (Fig. 5) to label each token to a specific type:

- *title*: Lines which follow capitalization rules for titles. We separate them from other lines in our process because titles never indicate an access control related requirement.
- *list start*: These lines represent the header or description of a list that follows.
- *list element*: These lines represent individual items contained within an ordered or unordered list. These lines are combined with the start of the list when sent to the parser and for classification. Combining the two provides additional context to both human analysts and machine classifiers.
- *normal sentence*: These lines represent statements that are not considered titles, list starts, or list elements.

document	→	*line*
line	→	*listID* title *line* | title *line* | *sentence line* | λ
sentence	→	normalSentence | listStart (":" | "-") *listElement*
listElement	→	*listID sentence listElement* | λ
listID	→	*listParanID* | *listDotID* | number
listParanID	→	"(" *id* ")" *listParanID* | *id* ")" *listParanID* | λ
listDotID	→	*id* "." *listDotID* | λ
id	→	letter | romanNumeral | number

Figure 5: Document Grammar

Further, we identify heading and list identifiers (e.g., "4.1.1" and "•"). The process removes these identifiers from the sentence passed into the natural language parser in Step 2 as the identifiers created issues with the generated output from the parser. If any irregularities are found, the parser defaults the current token (sentence) to "normal sentence" to recover.

Within Fig. 2, italicized words represent nonterminal symbols that can be replaced by other symbols on the right-hand side. Words in normal font are terminal symbols. Characters within quotation marks are also specific terminal symbols. λ represents an empty expansion of a nonterminal.

Step 2: Process Natural Language

After identifying the different sentence types, the process parses each line (sentence) with the Stanford Natural Language Parser (NLP) and outputs a graph in the Stanford Type Dependency Representation (STDR) [28]. While the Stanford Parser has several output formats available, we choose the STDR because it incorporates the sentence's syntactic information in a concise and usable format.

To replace shorthand or remove text that the parsing would not recognize, the process allows for a series of regular expressions to be applied to the text. Specifically in our work, we use this mechanism to replace 'w/' with 'with' and '/' with 'or.'

As the Stanford Parser processes sentences, it tags each word with a part of speech. Due to differences in text used to train the parser versus text used by our process, the parts of speech may be incorrect. To overcome this issue, we inserted a custom method into the parsing pipeline to override the part of speech tags if they are incorrect. For instance, we discovered that the parser always tagged "displays" as a plural noun whereas in most sentences in our text "displays" is a verb. The custom method looks for specific patterns among a group of words, and then replaces the incorrect part of speech. Incorrect tags can lead to an incorrect dependency graph being generated. In turn, an incorrect dependency graph can limit the effectiveness of the ACRE process. Both overrides are configurations established at the time the tool starts and are applied without user intervention to the entire text.

Fig. 6 demonstrates the produced STDR for the sentence "a nurse can order a lab procedure for a patient." Each vertex contains a word from the sentence along with that word's part of speech. In the figure, "DT" represents a determiner, "MD" a modal verb, "NN" a noun, and "VB" indicates a verb. Edges represent the grammatical relationship between two words. For instance, "nurse" functions as the nominal subject (nsubj) for "order" and "dobj" is the object to be ordered.

From the STDR generated by the parser, we create our sentence representation (SR) as ACRE needs to track additional attributes for the sentence and for each word. Additionally, some words in the original sentence are not required for our purposes and, hence, removed from the SR.

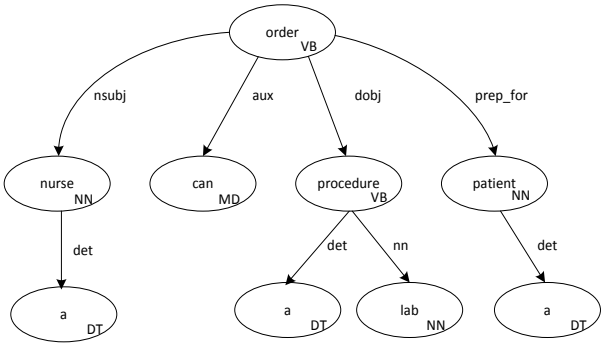

Figure 6: Stanford Collapsed Type Dependency Graph

Fig. 2 shows our corresponding SR for the same sentence as in Fig 6. The primary differences between the two graphs are the number of vertices and how each word is represented within a vertex. Within our SR, vertices correspond to words in the sentence and contain the word, the word's lemma[4], part of speech, domain flag, and access control policy indicators. The indicators correspond to the subject("S"), action("A"), resource("R") typically defined within an access control tuple.

Using a pre-order traversal, the process creates the SR from the Stanford graph. As each vertex is created, we make two changes to the nodes. First, to avoid multiple versions of the same word, we use the lemma of the original word. Second, to avoid differences in the part of speech, we collapse the parts of speeches for all nouns and verbs to their base category. For example, we treat all plural nouns and proper nouns as just nouns. Similarly, verbs with different tenses are treated collectively as a single group. We use a very small stop word list to remove common determiners[5] from the SR. Additionally, we check if it is feasible to collapse adjective and noun modifiers into parent noun nodes. Fig. 2 demonstrates this collapsing as we combined "lab" and "procedure". By removing extraneous nodes from the SR, we reduces the overall size of each graph, which in turn, provides fewer irrelevant attributes to a machine learning algorithm and provides for more concise patterns to be used in extracting access control policies.

Step 3: Classify Sentence as Access Control or Not

After Step 1 and Step 2 are both completed, a machine-learning algorithm classifies a sentence as access control or not. If the sentence does not express an access control policy, we perform no further analysis on it.

The process uses a *k*-NN classifier as the primary classifier. Such classifiers work by taking a majority vote of the existing classifications of the k nearest neighbors to the item under test. Thus, in our situation, to classify a sentence, the classifier needs to find which existing classified sentences are most similar to the current sentence under test. *k*-NN classifiers use a distance metric to find the closest neighbors. This metric is the sum of the differences among the attributes used to determine the classification. Typically, Euclidean distance serves as a metric for numerical values while for nominal values (e.g., words), the distance is generally considered to be zero if both values are the same or one if they differ. Our situation is more complex as we have a variable number of attributes (words, parts of speech, named entities) to consider for each sentence based upon the sentence length. Additionally, certain words may be more closely related to one another than other words. As such, we need to utilize a custom distance metric to compute a value representing the difference between two sentences.

Our distance metric is a modified version of Levenshtein distance [29]. Rather than using the resulting number of edits to transform one string into another as the value as the Levenshtein distance does, our metric computes the number of word transformations to change one sentence into another. Rather than strictly using just zero or one as the difference between words, the metric uses the function defined in Fig. 7. The function first checks the structure of graph around each vertex to ensure it corresponds to other vertex. Next, the function checks if the two vertices are the same (lemmas are equal). In line 7, we check if both words are numbers. Next, line 8 checks if both words are the same type of named entity such as a person or an organization. Then in line 9, the function checks if the two words are related through sets of cognitive synonyms (synsets)

[4] A lemma is a common root word for a group of words. For instance, sang, sung, and sings are all forms of a common lemma "sing." A stem is the root of a word after a suffix has been stripped. [7]

[5] a, an, the

within WordNet [6] via semantic relationships (hypernym or hyponym). If a relationship value is found, then a value between 0.1 and 0.4 is returned based upon the number of relationships traversed. Finally, a default value of 1 is returned if none of the other conditions are met.

```
computeVertexDistance(Vertex a, Vertex b)
 1: if a = NULL or b = NULL return 1
 2: if a.partOfSpeech <> b.partOfSpeech return 1
 3: if a.parentCount <> b.parentCount return 1
 4: for each parent in a.parents
 5: if not b.parents.contains(parent) return 1
 6: if a.lemma = b.lemma return 0
 7: if a and b are numbers, return 0
 8: if ner classes match, return 0
 9: wnValue = wordNetSynonyms(a.lemma,b.lemma)
10: if wnValue > 0 return wnValue
11: return 1
```

Figure 7: Compute Vertex Distance Logic

In prior work [30], we found that if we used a similarity threshold for the nearest neighbor(s) to determine whether or not to provide a classification answer, the k-NN classifier F_1 performance would be 1.0 (no misclassifications), although not all of the sentences would be classified. As such, we decide to utilize multiple machine learning algorithms to produce the final classification result. If the k-NN classifier's threshold is below a certain ratio (0.6) based upon the computed distance to the nearest neighbor(s) compared to the length of the sentence, we return the k-NN classifier's answer. Otherwise, we return a majority vote of the k-NN, naïve Bayes, and SVM classifiers. We term this classifier as "Combined SL."

Once the process makes determines if the sentence is related to access control or not, the user may review the determination and correct it if necessary within the tool. Fig. 10 shows a screenshot of the tool's user interface. The top table contains the document with individual columns to display the line number, sentence type, assigned classification, and completion status, assigned cluster (groups of similar sentences, optional functionality), and the sentence themselves. The dialog in the lower left allows users to review and manually enter or correct access control policies (discussed in the next section). The area in the lower right displays the SR.

Step 4: Extract Access Control Elements

Next, we need to extract the subject, action, and resource elements from the SR. We utilize a relation extraction approach for the identification of access control elements and subsequent extraction of the process. The approach follows a well-known bootstrapping technique [7], but has been adapted specifically for access control policy extraction.

To initialize the process (presented in Fig. 9), we seed a set of ten basic access control patterns with each pattern consisting of just three nodes as shown in Fig. 8. Each pattern is the same, except a different verb[7] is utilized for "Specific Action". Wildcards are used to match any nouns in sentences containing the pattern. We initially choose the words "create", "retrieve", "update", and "delete" because the words are commonly associated with viewing and manipulating data. We then examined the frequency of all verbs within the document and chose to add six more verbs associated with data and appearing with high frequencies within the document. Based upon the application domain or other documents, users may choose a different set of starting actions. From these patterns, we match all occurrences of the subjects and resources within the document along with their associate frequency counts. From the counts, we computed the mean values for the subjects and resources. We then assume any word that occurs more than the mean legitimately belongs to the application domain. Without a threshold, the potential for misidentified subjects and resources is much greater as any word matching the pattern would be accepted.

The subjects and resources are then stored in a listing of known subjects and resources. From this listing, we then search the document where any subject exists along with any resource. For each sentence that does match the condition, we extract the dependency pattern between subject and resource vertices. We then assume any verbs existing in that pattern are the actions. If more than one verb exists in the shortest path from the subject to the object, we combine the verbs. In the sentence, "the administrator chooses to create a new patient", we combine "choose" and "create"

[6] http://wordnet.princeton.edu/

[7] create, retrieve, update, delete, edit, view, modify, enter, choose, select

to "choose create" for the action. The subject would be "administrator" and the object would be "patient". We derive permissions for each pattern based finding the closest synonym in WordNet were a permission has already been defined in the process for an action.

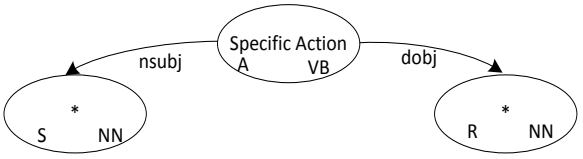

Figure 8: Basic Access Control Seed Pattern

Once we extract the pattern, we apply a series of transformations to extract additional patterns that may locate additional access control policies. Specifically, we transform patterns that have an active voice into passive voice and vice versa. We also transform the patterns to assume conjunctions may exist for two or more subjects, two or more actions, and two or more resources.

From the pattern set, we then search the documents for any sentences matching one or more patterns. Once we find any match, we check to see if other patterns match the same sentence. If more than one pattern does match and one pattern can be considered a "sub-pattern" of another pattern, we discard the "sub-pattern" match from the list of results as the other graph has provided a more specific match. Additionally, we check the matched sentences for any children nodes of the matched pattern that imply negativity or subject limitation (i.e., we look to see if there is relevant indicator just outside of the match subgraph).

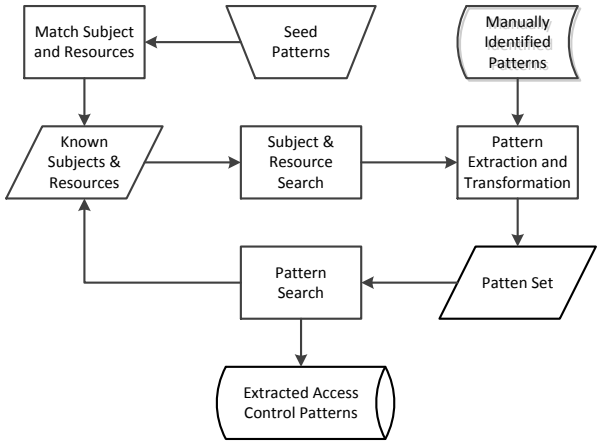

Figure 9: Access Control Extraction Overview

The extracted access policy is then stored in a list for validation and output to the user. Any new subjects or resources are then added to the list of known subjects and resources. If newly discovered subjects or resources exist, then the bootstrapping process can repeat until no new items or patterns are discovered. Once the process has stopped, the user may manually identify access control patterns. The information from these patterns is feed into the process to search for additional extracted elements.

Step 5: Validate Access Control
In this step, the tool checks for coverage and conflicts within the extracted access control policies. Coverage is reported as measure for each subject as to the number of identified resources that it has access control rules identified. As we assume a default of no-access, 100% coverage is not required. However, low coverage values may indicate a need for further access control policies. Conflicts occur within our process when a specific subject has been both granted permission to a specific resource and restricted for the same permission on the same resource. Such conflicts may arise due to policy extraction in multiple locations or the use of a limiter to restrict access to a specific subject.

V Evaluation Methodology

This section describes the application we utilized to evaluate our process, how the study oracle was created, and then how we performed the evaluation.

1 Application: iTrust

To evaluate the procedure, we used iTrust as our test system. iTrust, a web-based healthcare application originated as a class project for Software Engineering at North Carolina State University in 2004, and has been enhanced by classes each semester through 2013. The application follows a typical three-tiered architecture with logical layers for the presentation, application, and persistence. Instructors, teaching assistants, and students have contributed to the application, which is currently in its 15th version. Each class performs software enhancement and maintenance on the application, correcting defects and implementing new functionality. The requirements consist of 40 use

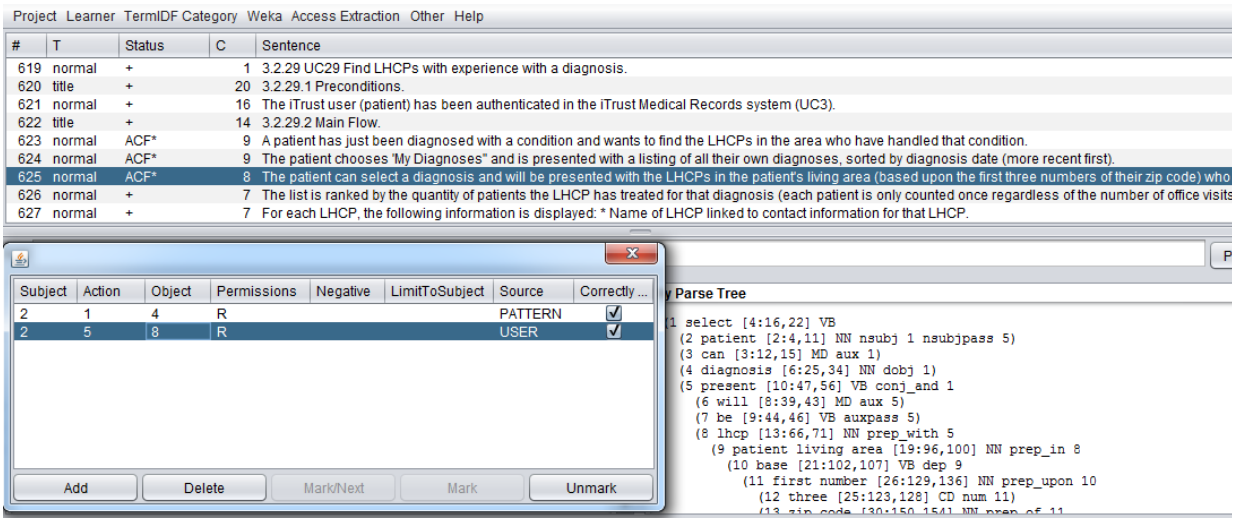

Figure 10: Access Control Relation Extraction Tool Screenshot

cases plus additional non-functional requirements, constraints, and a glossary. The version we used contained 1159 sentences with 409 (36.7%) of those sentences classified as containing one or more access control policies.

2 Study Oracle

First we created our oracle in which we manually classified each statement in the iTrust Requirements Specification. We first converted the document into a text-only format. The only changes made to the resulting text file were to account for misplaced line breaks and to remove tables since our process cannot evaluate information in a table-based format. (Tables consisting primarily of sentence-based content were retained.) Next, we imported the document into ACRE Tool so we can classify each sentence. After the initial import, each sentence has been parsed and converted into our SR. The first authors classified the 1,159 sentences (or lines) in seven hours.

After the initial classification was completed, we validated the classification through several approaches. First, we used a k-medoids clustering algorithm to compute clusters of related sentences. We then compared the classifications within each cluster. Ideally, every sentence should be marked the same. We investigated further those sentences that did not have the same classification as other sentences in the group. Additionally, as we classified each sentence, we had access to the neighbors contained within the k-NN classifier. This approach allowed for more rapid manual classification by suggesting initial classification that we could then verify or correct as deemed necessary. Additionally, any discrepancies in the predicted classification could be easily traced back to the source sentences where the appropriate change could be made.

Next, the first author then manually identified the access control policies contained in the 409 sentences marked as requiring access control. Each access control policy had all relevant elements (subject, action, resource, etc.) identified. This effort took 12 hours.

3 Study Procedure

Once the oracle has been created, we executed five classifiers (the k-NN classifier, a TF-IDF classifier, the "Combined SL" classifier, a multinomial naïve Bayes classifier and a SVM - sequential minimal optimization classifier) on the requirements document. We utilized a variety of classifiers to measure performance differences across the different algorithms. For each classifier considered, we tested using a stratified n-fold cross-validation and computed the precision, recall, and F_1 measure. With the n-fold cross-validation, data is randomly partitioned into n folds based upon each fold of approximately equal size and equal response classification. For each fold, the classifiers are trained on the remaining folds and then the contents of the fold are used to test the classifier. The n results are then averaged to produce a single result. We follow Han et al.'s

recommendation [31] and use 10 as the value for n as this produces relatively low bias and variance. The cross-validation ensures that all sentences are used for training and that each sentence is tested just once. In addition to our versions of the k-NN classifier and TF-IDF classifier, we utilized the multinomial naïve Bayes and SVM - sequential minimal optimization classifiers within the Weka [32] suite. We directly accessed the Weka classifiers through the available Java APIs. As the folds are randomly generated, we executed the tests 3 times and averaged the results.

In the final phase of the study, we examined seeding the process with different sets of initial actions (verbs). From the patterns generated, we extracted the access control policies from the requirements document and compared the extracted to the manually identified policies.

VI Experimental Results

This section presents our evaluation of the research questions.

RQ1: How effectively can we identify access control policies in natural language text in terms of precision and recall?

Fig. 11 presents the results of executing each classifier against the entire document set using a ten-fold cross validation. We executed each test three times and present the average.

			F_1
Naïve Bayes	.743	.940	.830
SMO	.845	.830	.837
TF-IDF	.588	.995	.739
k-NN (k=1)	.851	.830	.840
Combined SL	.873	.908	.890

Figure 11 Stratified Ten-Fold Cross Validation

Creating the "Combined SL" classifier did produce some performance gains from using individual classifiers as the F_1 Measure was .05 higher than the next best performer (k-NN, k=1). For the k-NN classifier, we did experiment with various values for k and found one produced the best performance.

RQ2: What common patterns exist in sentences expressing access control policies?

By examining the most frequently occurring patterns from our manual identification, we found that the basic pattern displayed in Fig 5. occurred in 25% of the sentences marked for access control. This occurrence doesn't require a small sentence, but rather somewhere in the sentence we found three nodes and two edges of that pattern. Another frequently occurring pattern (8%) occurs with sentences start with "The *subject* [chooses|selects] to *perform action.*" A wide range of patterns existed due to differences in prepositions represented on edges.

RQ3: What is an appropriate set of seeded graphs to effectively bootstrap the process to extract the access control elements?

To evaluate this question, we started the process with different sets of base words and then compared the quantity of patterns generated and the performance of those patterns to extract the correct access control statements from the sentence. The best performance came with the set of 10 action verbs defined for the seed with a precision of .463 and a recall of .536.

VII Limitations

Several limitations exist within this work. As ACRE utilizes NLP techniques, the process and associated tool cannot extract information contained in images. With regards to access control policies, our bootstrapping approach does not take into account the presence of contextual information or conditions that may affect the generated access control. The user can manually enter such information, though. The approach also requires that subjects and resources be identified as nouns and actions as verbs unless the user manually enters a policy. We also assume all necessary information for an access control policy is contained within the same sentence. It is feasible for elements either to exist in surrounding sentences. We also have not handled resolution issues at this time. These issues occur when a pronoun or generic term such as "system" or "data" is used in place of a descriptive term. Our work has a significant external validity threat as we examined only one document for one system in a specific problem domain. While the process does

not have any specific problem domain constraints, additional evaluation needs to occur across multiple domains and applications. We surmise that the process will work for other narrative based texts, but "task/step oriented" documents such as test scripts and user manuals would be less effective as the subject is often assumed throughout a series of steps. For such documents, we would need to investigate the use of "action – resource" pairs to generate patterns.

An internal validity threat may exist as the first author performed all of the sentence classifications and access control policies. To check the accuracy of the first author's classifications, we had five software developers classify a representative sample of 30 sentences. Utilizing Randolph's Online Kappa Calculator [33], we calculate a free-marginal kappa of 0.86 (indicating significant inter-rater agreement) by comparing the first authors classification against the majority vote of the other raters.

VIII Future Work

For future work, we plan to continue work on the tool to resolve the resolution issues presented in the previous section. We should also be able to detect and report missing elements and other issues directly to users. We also plan to develop a much larger corpus of text documents for multiple systems in two or three domains. Utilizing the corpus, we can more effectively measure how the process performs and would generalize to other systems and problem domains. Other planned work involves extracting more complicated access control policy such as that required for privacy-based controls or access based upon specific contexts such as time or location. We also look to create other derivative patterns to increase the recall while adding checks to improve precision.

IX Conclusion

In this paper, we present a new process, ACRE, and tool that assist developers in automatically extracting access control policies from natural language text. The tool provides a mechanism for developers to quickly generate an initial set of access control policies with traceability back to the originating set. Developers can utilize the process to detect conflicts in generated policies as well as evaluate the coverage of generated policies to the identified subjects and resources. We demonstrated how effective a bootstrapping process can extract policies from a very small initial set of patterns. We also presented a grammar that can be applied when parsing text documents to provide additional context information for specific elements within the document.

As we utilized a cross-fold validation on a single document, our combined classifier showed very effective performance with a F 1 Measure of .89. We also showed improved performance through combine the results of multiple classifiers. However, our performance in extracting access control patterns was substantially less with a precision of .463 and a recall of .536.

Acknowledgment

This work was supported by the U.S. Army Research Office (ARO) under grant W911NF-08-1-0105 managed by NCSU Secure Open Systems Initiative (SOSI). We would like to thank the North Carolina State University Realsearch group for their helpful comments on the paper.

References

[1] J. Bedard, R. Hoitash, U. Hoitash, and K. Westermann, "Material Weakness Remediation and Earnings Quality: A Detailed Examination by Type of Control Deficiency," *Auditing: A Journal of Practice & Theory*, 2012.

[2] "2011 CWE/SANS Top 25 Most Dangerous Software Errors," 2011. [Online]. Available: http://cwe.mitre.org/top25/. [Accessed: 14-Nov-2011].

[3] Verizon RISK Team, "2013 Data Breach Investigations Report," 2013.

[4] M. Mazzetti and M. Schmidt, "Ex-Worker at C.I.A. Says He Leaked Data on Surveillance," *New York Times*, New York, NY, USA, 09-Jun-2013.

[5] P. Samarati and S. de Vimercati, "Access control: Policies, models, and mechanisms," *Foundations of Security Analysis and Design*, pp. 137–196, 2001.

[6] "Identity & Information Assurance-Related Policies and Issuances," 2012. [Online].

Available: http://iac.dtic.mil/iatac/download/ia_policychart.pdf. [Accessed: 01-Oct-2012].

[7] D. Jurafsky and J. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, Second. Pearson, 2009, p. 988.

[8] A. Meneely, B. Smith, and L. Williams, "iTrust Electronic Health Care System: A Case Study," in in *Software System Traceability*, 2011.

[9] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," *Machine Learning: ECML-98*, 1998.

[10] N. Chinchor and B. Sundheim, "Message Understanding Conference - 6: A Brief History," in *Proceedings of the 16th conference on Computational Linguistics - Volume 1*, 1996, pp. 466–471.

[11] J. Piskorski and R. Yangarber, "Information Extraction: Past, Present, and Future," in in *Multı-source, Multilingual Information Extraction and Summarization*, T. Poibeau, H. Saggion, J. Piskorski, and R. Yangarber, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 23–50.

[12] Q. He and A. I. Antón, "Requirements-based Access Control Analysis and Policy Specification (ReCAPS)," *Information and Software Technology*, vol. 51, no. 6, pp. 993–1009, Jun. 2009.

[13] X. Xiao, A. Paradkar, S. Thummalapenta, and T. Xie, "Automated Extraction of Security Policies from Natural-Language Software Documents," in *International Symposium on the Foundations of Software Engineering (FSE)*, 2012.

[14] R. Schwitter, "Controlled Natural Languages for Knowledge Representation," in *Proceedings of the 23rd International Conference on Computational Linguistics*, 2010, pp. 1113–1121.

[15] C. a. Brodie, C.-M. Karat, and J. Karat, "An Empirical Study of Natural Language Parsing of Privacy Policy Rules Using the SPARCLE Policy Workbench," *Proceedings of the second symposium on Usable privacy and security - SOUPS '06*, p. 8, 2006.

[16] P. Inglesant, M. A. Sasse, D. Chadwick, and L. L. Shi, "Expressions of Expertness: The Virtuous Circle of Natural Language for Access Control Policy Specification," in *Proceedings of the 4th symposium on Usable privacy and security*, 2008, pp. 77–88.

[17] L. Shi and D. Chadwick, "A Controlled Natural Language Interface for Authoring Access Control Policies," in *Proceedings of the 2011 ACM Symposium on Applied Computing*, 2011, pp. 1524–1530.

[18] M. Hearst, "Automatic acquisition of hyponyms from large text corpora," in *Proceedings of the 14th conference on Computational Linguistics*, 1992, pp. 539–545.

[19] R. Snow, D. Jurafsky, and A. Y. Ng, "Learning Syntactic Patterns for Automatic Hypernym Discovery," in *Advances in Neural Information Processing Systems 17*, 2004, vol. 17, pp. 1297–1304.

[20] A. Akbik and J. Broß, "Wanderlust: Extracting semantic relations from natural language text using dependency grammar patterns," in *Workshop on Semantic Search*, 2009, vol. 491.

[21] G. Zhou, J. Su, J. Zhang, and M. Zhang, "Exploring Various Knowledge in Relation Extraction," in *Proceedings of the 43rd Annual Meeting of the ACL,*, 2005, no. June, pp. 427–434.

[22] K. Fundel, R. Küffner, and R. Zimmer, "RelEx--relation extraction using dependency parse trees.," *Bioinformatics (Oxford, England)*, vol. 23, no. 3, pp. 365–71, Feb. 2007.

[23] M. Mintz, S. Bills, R. Snow, and D. Jurafsky, "Distant supervision for relation extraction without labeled data," in *Proceedings of the 47th Annual Meeting of the ACL and the 4th IJCNLP of the AFNLP*, 2009, no. 2005, pp. 1003–1011.

[24] F. Wu and D. Weld, "Open information extraction using Wikipedia," in *Proceedings of the 48th Annual Meeting of the Association for*

Computational Linguistics, 2010, no. July, pp. 118–127.

[25] Mausam, M. Schmitz, R. Bart, S. Soderland, and O. Etzioni, "Open language learning for information extraction," *EMNLP-CoNLL '12 Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pp. 523–534, 2012.

[26] R. M. W. Dixon, *A Semantic Approach to English Grammar*, Second. Oxford University Press, USA, 2005, p. 543.

[27] R. Huddleston and G. Pullman, *The Cambridge Grammar of the English Language*, First. Cambridge University Press, 2002, p. 1860.

[28] M.-C. de Marneffe, B. MacCartney, and C. Manning, "Generating Typed Dependency Parses from Phrase Structure Parses," *Proceedings of Language Resources and Evaluation*, pp. 449–454, 2006.

[29] V. I. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.

[30] J. Slankas and L. Williams, "Classifying Natural Language Sentences for Policy," *2012 IEEE International Symposium on Policies for Distributed Systems and Networks*, pp. 33–36, Jul. 2012.

[31] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed. Morgan Kaufmann, 2011, p. 744.

[32] M. Hall, H. National, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software : An Update," *SIGKDD Explorations*, vol. 11, no. 1, pp. 10–18, 2009.

[33] J. J. Randolph, "Online Kappa Calculator," 2008. [Online]. Available: http://justusrandolph.net/kappa/. [Accessed: 02-Apr-2013].