Extracting Database Role Based Access Control from Unconstrained Natural Language Text

John Slankas and Laurie Williams
North Carolina State University
890 Oval Drive
Raleigh, NC 27695-8206, USA
+1(919) 515-7926
[john.slankas,laurie_williams]@ncsu.edu

ABSTRACT
Although software can and does implement access control at the application layer, failure to enforce data access at the persistence layer often allows uncontrolled data access when individuals bypass application controls. *The goal of our research is to help developers improve security by ensuring the access controls defined within unaltered natural language texts are appropriately implemented within a system's persistence layer.* Access control implemented in both the application and persistence layers strongly supports a defense in depth strategy. We propose a tool-based process to 1) parse existing, unaltered natural language documents such as requirements and policy statements; 2) classify a statement as being access control related or not; and, as appropriate, 3) extract subjects, actions, and data objects; and 4) automatically generate the necessary SQL commands to enforce role-based access control within a relational database. To evaluate our process, we analyzed a public requirements specification. Our classifier correctly predicted 78% of the access control statements while recalling 92% of the applicable statements. The process correctly identified and mapped 92% of the actual database tables to the extracted resources. We successfully executed the application with application and persistence layer role based access control in place.

Categories and Subject Descriptors
K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms
Documentation, Reliability, Security, Verification.

Keywords
Security, persistence, access control, role based access control, RBAC, classification, natural language parsing

CODASPY'13, February 18–20, 2012, San Antonio, Texas, USA.

1. INTRODUCTION
In the 2011 CWE/SANS Top 25 Most Dangerous Software Errors [1], 30% of the errors directly relate to access control[1]. Data breaches continually make news headlines such as when Israeli authorities charged a contract worker from Labor and Welfare Ministry with stealing the personal information of over nine million Israelis from the country's Population Registry [7] in October 2011.

To mitigate data security issues, organizations must properly implement access control across all system components. Developers often utilize role-based access control (RBAC), considered the "gold standard" [10], to enforce authorization policies within systems. However, many software systems establish the access control only at the application level. If those controls are breached due to malicious activity or defective implementation, no additional layers of security exist to protect data within the database. Insider threats exist if individuals can bypass application level security and access the database directly. Additionally, programmers can intentionally or inadvertently implement functionality that violates the RBAC policy. A defense in depth security approach requires access control at *both* the application and persistence layers. However, creating and defining RBAC policy for databases can be a tedious, time-consuming, and error-prone endeavor. Developers must sift through existing documentation, application code, and database implementations.

The goal of our research is to help developers improve security by ensuring the access controls defined within unaltered natural language texts are appropriately implemented within a system's persistence layer.

We propose a process, which we call Role Extraction and Database Enforcement (REDE), which allows organizations to utilize existing, unconstrained natural language text to generate RBAC policies for databases. REDE analyzes requirements or other natural language statements for their subject, action, and resource parts. REDE then uses an instance-based learner to make a

[1] Including CWE-306,862,798,829,250,863,732,307, and 807.

decision about whether the sentence is related to access control or not. If the sentence involves access control, then the process extracts actors, permissions, and data objects, maps the data objects to existing database tables, and then generates the appropriate SQL commands to execute within a database to enforce the appropriate access control. We evaluated our process and tool against 1114 statements from the requirements for an open source educational testbed, the iTrust[2] Electronic Health Records System.

The contributions of this paper are as follows:

- We introduce the first methodology to classify natural language statements as RBAC policy-related or not.
- We present a process and a supporting tool that can assist developers and security experts in automatically generating SQL commands to create RBAC within a DBMS.
- We present a distance function to compute the similarity between two sentences.

The rest of this paper is organized as follows: Section 2 provides requisite background information Section 3 presents related work. Next, Section 4 details REDE and the associated tool. Section 5 presents our evaluation of the approach. Section 6 discusses limitations. In Section 7, we discuss future work. Finally, we conclude in Section 8.

2. BACKGROUND

In this section we provide background with regards to access control, natural language parsing, machine classification, and evaluation criteria.

2.1 Access Control

Access control is a critical application mechanism to ensure confidentiality and integrity [27]. Most access control models utilize a tuple *(subject, resource, action)* to represent a rule as to whether or not the *subject* (a user) can perform the requested *action* on the specified *resource* (object) within the system. Access control models are grouped into three classes: Discretionary Access Control (DAC), Mandatory Access Control (MAC), and Role-Based Access Control (RBAC). All three classes control access based upon the subject's identity and what permissions they have been granted. They differ in how the access is granted. In DAC, the resource's owners have the authority to decide who can access the resource. In MAC, organizational rules determine who can access data. Military and intelligence systems often utilize MAC to ensure users have the appropriate security clearances to access data. Finally, RBAC controls access based upon rules granted to roles with users assed to roles. Most relational database management systems (RDBMS) support forms of DAC and RBAC[3]. Roles were added to the 1999 Structured Query Language Standard (SQL) [31].

2.2 Classification

To determine whether a sentence implies access control or not, we classify each sentence using machine learning techniques. Specifically, we employ a *k*-nearest neighbor classifier (*k*-NN) which works by classifying a test item based upon which items are classified as closest to 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 generally considered to be zero if both attribute values are the same or one if they differ. *k*-NN classifiers may use custom distance functions. Advantages of *k*-NN classifiers include ability to incrementally learn as new items are classified, to classify multiple types of data, and to handle large number of attributes for each item. The primary drawback to *k*-NN classifiers is that if they have *n* items stored, classification takes $O(n)$ time. Additionally, irrelevant attributes can skew classification results.

To train the classifier, we use an active learning approach [12]. As new items are evaluated by the classifier, we validate the initial response, correct the response if necessary, and then place the correctly classified item into the classifier.

2.3 Evaluation Criteria

To evaluate our classifier, we examine how well it produces correct results. As our classifier produces binary results, it either correctly or incorrectly predicts each class. We term correct predictions as true positives (TP) and true negatives (TN) if the classifier correctly predicted yes or no respectively. For incorrect classifications, we term these as false positives (FP) and false negatives (FN). To evaluate the binary classification, we use precision, recall, and the F_1 Measure. Precision (P) is the proportion of correctly predicted access control statements: $P = TP/(TP + FP)$. Recall (R) is the proportion of access control statements found: $R = TP/(TP + FN)$. The F_1 Measure is the harmonic mean of precision and recall, giving an equal weight to both elements: $F_1 = 2 \times \frac{P \times R}{P + R}$. 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. For example, a precision of .75 implies 75% of the access control statements are correctly generated, while 25% of the statements would grant additional access not required. A lower recall value implies that we will have missed access

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

[3] The notable exception is MySQL which does not provide support for RBAC. http://dev.mysql.com/doc/refman/5.6/

control sentences. So a recall of .75 implies that 75% of all of the actual access control statements needed would be discovered, leaving undefined the other 25%. Assuming access control starts from a basis of no access, having lower recall is not as significant of a problem from a security perspective, but would require additional verification to ensure all access is eventually granted.

3. RELATED WORK

This section reviews the work related to our proposed process.

3.1 Natural Language and Access Control

Other researchers have explored using natural language to generate access control policies from natural language. He and Antón [14] 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, Xaio et al. [30] present an approach, Text2Policy, where they parsed use cases to create eXtensible Access Control Markup Language[4] (XACML) policies. Their approach was specific to use cases and relied upon matching four specific sentence patterns to deduce the necessary information to populate an access control method. Using XACML as a target simplified their process as well as they did not have to map objects in the natural language to existing application elements or database entities. Our processing utilizes machine learning to make decisions and implements access control against a system's database.

3.2 Controlled Natural Language

Other researchers have resolved converting natural language to and from policies by utilizing a controlled natural language (CNL). Schwitter [28] 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. Policy authored outside of tools must confirm to strict limited grammars to be automatically parsed as well. Brodie et al. [4] 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 natural language into formal policy. Users also responded favorably to their policy authoring tool. Inglesant et al. [16] demonstrated similar success with their tool, PERMIS, which utilized 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 [29] 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 didn't support conditions or obligations. Our process removes the CNL constraint, working against original, unaltered texts.

3.3 Schema and Ontology Matching

A challenge with our approach is to map resources found in the natural language text to database tables and columns. This problem is similar to the challenges of matching one database scheme to another database schema as well as matching different ontologies. Multiple survey papers have been written on database schema matching [3, 9, 26] and ontology matching [6, 18]. Research continues in this field with such recent work as Po and Sorrentino [24] who utilized probabilistic techniques to derive lexical relationships and disambiguate word sense across different ontologies. Currently, our approach utilizes an approach based upon word similarity.

3.4 Databases from Natural Language

Different approaches exist to discover database entities from natural language. The first approach has been to produce database design from the natural language text. In 1983, Chen [5] first presented a series of heuristics to produce entity relationship diagrams from natural language. Hartman and Link [13] updated this work in 2007 through additional heuristics and refinements. Both approaches require humans to analyze the text. Omar [23] applied semantic role labeling techniques to automatically generated database design from a series of heuristics. Another approach [2, 25] has been with research necessary to utilize natural language to query databases. To effective perform natural language queries, a system must first comprehend the user's request and determine the specific data requested as well as any conditions for retrieving the data. Then the system must map the requested data elements and conditions to the physical database to generate the necessary query request by the user. Our process uses identified resources from the natural language text to map to existing database tables.

4. ROLE EXTRACTION AND DATABASE ENFORCEMENT

This section details our process, Role Extraction and Database Enforcement (REDE), to extract roles and database objects from natural language text and generate RBAC policies for applications that utilize relational

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

databases. To guide users through the process, we developed a tool to perform many of the tasks involved.

4.1 Overview

For input, the process takes natural language text, a physical database schema, existing database role names, and, optionally, a list of domain-specific words. Any number of existing project documents such as requirements, use cases, designs, test scripts, and training material can serve as the natural language text. The physical database schema consists of the existing table definitions. Database designs may be substituted if the physical implementation is unavailable. The process outputs the SQL commands to establish role-based access control in a database. Additionally, the process can generate consistency and traceability reports.

The REDE process consists of four primary steps:

1. Parse natural language. In this step, sentences are converted into an internal representation and analyzed for previously discovered elements and for negativity.
2. Classify sentence attributes. Next, we use a classifier to examine whether if the sentence relates to access control. The user may correct classifications.
3. Extract access control elements. The tool can find previously used subjects, actions, and objects based upon the most similar sentence or from previously found words. If the sentence has been identified as access control the user will need to ensure the extracted elements are correct.
4. Generate SQL Commands. Based upon the extracted access control elements, the system generates the necessary commands to generate role-based access control within a relational database.

4.2 Step 1: Parse Natural Language

The process begins by entering the text into the system, parsing the text and converting the parsed representation into REDE's internal representation (IR). Next, REDE performs three analyses of the IR and assigns additional attributes necessary for classification and to establish access control as the step's output.

4.2.1 Parse Text

First, the tool parses a paragraph at a time from the natural language utilizing the Stanford Natural Language Parser[5]. By breaking the input into paragraphs, we achieve a balance between providing an appropriate amount of text to the parser versus too little. While a sentence at a time could be processed, references among sentences (such as what noun a pronoun refers to) would not be tracked. Memory constraints prevent the parser from processing a complete document or even larger sections. For each sentence in the paragraph, the parser outputs a graph in the Stanford Type Dependency Representation (STDR) [21]. While the parser has several output formats available, we choose the STDR as it incorporates the sentence's structural 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. Both overrides are configurations established at the time the tool starts and are applied to the entire text.

4.2.2 Internal representation

From the STDR generated by the parser, we create our IR as REDE 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 IR. Figure 1 shows the 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 "create" and "dobj" is the object to be ordered. A sentence's primary verb typically forms the graph's "root."

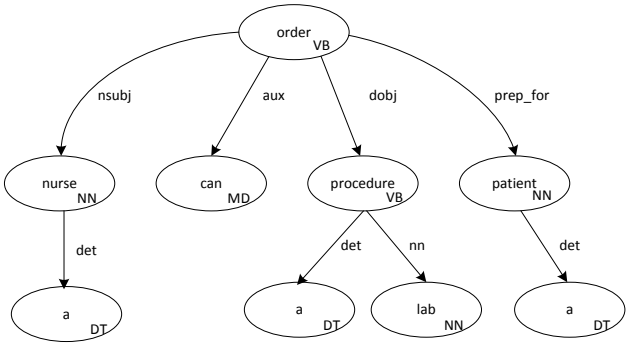

Figure 1: Stanford Collapsed Type Dependency Graph

Figure 2 shows our corresponding IR for the same sentence. The primary differences between the two graphs are the number of vertices and how each word is represented within a vertex.

[5] http://nlp.stanford.edu/software/

Within our IR, vertices correspond to words in the sentence and contain the word, the word's lemma[6], part of speech, domain flag, and database indicators. The domain flags correspond to the subject("S"), action("A"), resource("R") typically defined within an access control tuple. Edges are unchanged from the STDR.

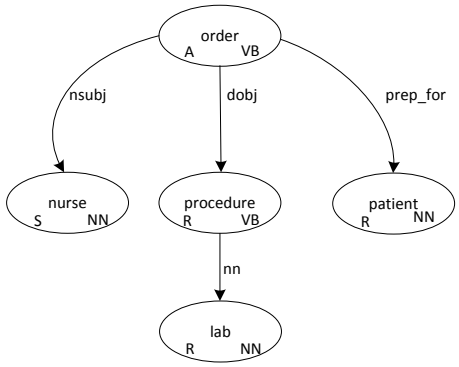

Figure 2: REDE Internal Representation

The process uses the method outlined in Figure 3 to generate the IR from the Stanford graph. The most significant change is to remove vertices that do not provide any semantic meaning from an access control perspective. (Vertices with children are not removed.) We consider two categories of extraneous words: articles and modal verbs. Articles are the words "a", "an", "the." While in the context of a sentence, these determiners may provide information to the reader, from an access control point they do not provide additional details as the access the modified noun remains unaffected by their use. Modal verbs include such items as "can", "shall", "may." While modal verbs may indicate conditions or obligations that need to be met, for database access control they are irrelevant as the user must have the access defined otherwise within the sentence. Removing these extraneous nodes from the IR reduces the overall size of each graph, which in turn, provides fewer irrelevant attributes to a machine learning algorithm used later in the process.

```
IRNode createIntRepresentation(STDRNode node) {
 1: if node.part_of_speech=article or modal verb
 2:   return null
 3: if node in visited list, return null
 4: add node to visited list
 5: IRNode irn = new IRNode(lemma(node.word),
 6:                         basePOS(node.POS))
 7: foreach child of node
 8:   irChild = createIntRepresentation(child)
 9:   if irChild = null, continue
10:   create edge from irn to irChild using the
11:         existing relationship
12: return irn
```

Figure 3: Create Internal Representation Algorithm

[6] 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. [17]

As part of a later step in the process, we compare sentences to each other to measure their similarity. To avoid spurious differences, we make two modifications to each vertex. First, to avoid multiple versions of the same word, we use the lemma of the original word within the node, which the Stanford parser provides as part of its output. Using the lemma provides increased accuracy over the use of a word's stem. Stemming algorithms are rule-drive approaches to derive base forms of words without taking into account the word's context or part of speech. In contrast, lemmatization takes into account the word's part of speech and other information sources such as a vocabulary to derive the base form of a word [20]. Second, to avoid differences in the part of speech, we collapse the parts of speeches for all nouns, verbs, and adjectives to their base category. For example, we treat all plural nouns and proper nouns as simple nouns. Similarly, verbs with different tenses are treated collectively as a single group.

4.2.3 Domain Dictionary and Lookup

With the IR in place, the process examines the graph to see if any of the words have previously been detected in prior sentences from an access control perspective. Previously used words in part of an access control tuple are marked with a domain flag for the subject, action, or resource.

The tool maintains a domain dictionary consisting of four entry types. The first entry type is an action verb, which represents an action the user takes within the application. For each action verb, we track the corresponding database permission (delete, insert, update, or select) required to perform the action within the database. For example, the "edit" verb has the permissions "select" and "update." The "select" permission is required as the user must have access to view the record and then "update" to modify the record in the database. We also track whether or not the verb implies a negative permission, which used to revoke the corresponding access from a role. Subjects form the second entry type in the dictionary. Subjects correspond to database roles. The third entry type is a resource, which corresponds to a table or column within the database. The fourth word type is a list of domain words relevant to the current application domain. Since the user provides these domain words, they are populated into the dictionary at the start of the process. Domain ontologies, taxonomies, and thesauri may serve as sources for these words.

The domain dictionary is pre-defined with eight verbs that correspond with actions such as create, retrieve, update, and delete. Additionally, the dictionary includes six verbs such as prohibit and stop that imply negative permissions as they designate permissions that should be revoked from a user. The domain dictionary exists independently of any text of project. As such, dictionaries unique to a given application domain, may be re-utilized across multiple systems or shared with other groups. As sentences are processed, new domain words will be added to the

dictionary. Synonyms may be defined by a user when they are two terms that refer to exactly the same entry. Admin and administrator is an example.

As the process looks up words from a sentence in the domain dictionary, a greedy approach is used to find the longest possible entry. Thus, if a sentence contains the words "health care representative" and the dictionary has both "health care representative" and "representative" defined, the system only matches "health care representative." The search process also takes into account the part of speech of a word as well. Subjects and resources must both have a noun as the last word, while actions must have a verb as the last word. The system does not use the relative position of a word within a sentence to perform dictionary searches.

If new words need to be added to the dictionary domain then the user may do so through the tool at any point after the IR has been generated and before the SQL commands are generated. From the sentence "A nurse can order a lab procedure for a patient," a user would add nurse as a subject, "order" as an action with an "insert" permission, and both lab procedure and patient as resources into the domain dictionary.

4.2.4 Negation

Denying users access permission to an object when they should not have access to that object is a critical feature of any access control system. As such, the process analyzes sentences to determine when permission should be explicitly revoked from a role. From the Dixon's negation concepts [8], we utilize three methods to detect negation within a sentence. First, we search the graph for edges for negative modifiers. Next, the process searches vertices for negative determiners (no, zero, neither), adjectives(unable) and nouns (none, nothing) [15]. Finally, the process checks verbs negative connotation (as expressed in the domain dictionary) or by a pre-determined list of negative prefixes associated with English words. If multiple negations are detected, the process displays a message to the user and the negativity flag remains unchanged. If the process incorrectly sets the negativity flag, the user may explicitly set the value. With the sentence "Nurses may not write prescriptions," the tool would set the negative flag as nurses should not have the privilege to write prescriptions for any patients.

4.2.5 Restrictive Focus Modifiers

Within a system, privileges are often limited to one or more specific roles. For example, only an administrator can maintain system lookup tables. In this scenario, we want to grant permission to the administrator role while restricting the permission from other roles. Within English, *restrictive focus modifiers* [15] express such limitations. As the position of such modifiers (i.e. 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," our process would grant the role "doctors" the insert capability on the table "prescriptions." 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 only set the only flag for a sentence if the modifier applies to the subject identified in the sentence. Users may override the flag if the process set it incorrectly.

4.3 Step 2: Classify Sentence Attributes

Once the tool completes the parsing and initial analysis of a sentence, an instance based algorithm classifies additional sentence attributes. The primary classification is whether or not the sentence implies access control. The tool also classifies words as belonging to access control elements if the domain dictionary lookup did not already discover the elements. The access control classification is paramount to our process as the classification of "yes" indicates the process will generate an access control statement.

To classify sentences, REDE utilizes a k-NN classifier with a custom distance metric. As a k-NN classifier works by taking a majority vote of the existing classifications of the k nearest neighbors, the classifier uses a distance metric to find those neighbors. In this process, to classify a sentence as access control, the classifier needs to find which sentences already classified are most similar to that sentence. To calculate the required distance metric between two sentences, the process performs a recursive, pre-order traversal of each sentences' IR and sums the distances between vertices in one graph compared to their equivalent vertices in the other graph. To calculate the distance for each vertex, the process uses the function presented in Figure 4. The different values were chosen as way to express the difference between two nodes. If the nodes are the same, then a value of 0 is returned. If they nodes are different, then a value of 1 is returned. However, there are situations in which nodes are similar, but not exactly the same. As such, we return a value in between 0 and 1 to represent the closeness. In line 7, the process checks to see if vertices share the same domain flag which occurs when the domain dictionary contains the lemma from both vertices. In this situation, the graph has found the domain flag in equivalent locations and a small value is returned. (The value ranges from 0.1 to 0.4 depending upon the structure of the sub-graphs rooted at a and b. In line 9, the process checks to see if the two words are related through sets of cognitive synonyms (synsets) within

WordNet[7] 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. Next, the process checks two see if both lemmas are contained within the list of domain terms provided as input to the overall process. If both words are found, a value of 0.5 is returned. Finally, a default value of 0.75 is returned if none of the other conditions are met. In this situation, the vertices have an equivalent structure and part of speech and should be scored as closer together than two vertices differing in those attributes. We choose these values as they maximized the classifier's precision with regards to the requirements document utilized.

```
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: dfValue = domainDistance(a,b)
 8: if dfValue > 0 return dfValue
 9: wnValue = wordNetSynonyms(a.lemma,b.lemma)
10: if wnValue > 0 return wnValue
11: if domainKnowledgeChk (a.lemma,b.lemma)
      return 0.5
12: return 0.75
```

Figure 4: Compute Vertex Distance

While the presence of domain flags for subjects, resources, and actions may be used as an alternative to a machine learning classifier. The presence or absence of domain flags is not sufficient to appropriately classify a sentence as containing to access control or not. First, it may be possible for the three different domain flags to be present in various parts of the sentence, but not necessarily in locations that indicate database access is necessary. Second, in many sentences may not contain three different domain flags, yet the sentence does require data access.

Once the classification is complete, the user may review the classifications and provide any corrections as necessary through the tool.

4.4 Step 3: Extract Access Control Elements

Next, we need to extract the subject, action, and resource elements from the IR. In addition to the database lookup in section 4.2.3, we also can utilize the results of the nearest neighbor and use the position of the elements in that sentence for the elements in the current sentence. For the sentence, "A nurse can order a lab procedure for a patient," the subject would be nurse, the action is order, and the resource is patient and lab procedure.

Another situation arises in which the sentence has been classified as access control, but portions of the access control tuple are not directly present in the sentence. Examples include sentences that provide additional details for the preceding sentences or generic terms such as "system", "record", or "data." The natural language processing field refers to this problem as reference resolution [17]. REDE performs resolution by utilizing the last occurrence of the missing tuple. Our tool highlights marks this resolution and allows the user to change the tuple value if necessary.

Next, we need to map the discovered resources and objects to roles and tables in the database. Through the tool, users must manually match up each subject and resource with a role or table. For each subject and resource entry in the domain dictionary, the tool provides a sorted list of possible choices for the user to select. The choices are sorted by the Levenshtein distance [19] between the current subject or entry being mapped to the possible choice from the database. We choose to use the Levenshtein distance as it is a commonly used algorithm to measure the similarity between two strings.

4.5 Step 4: Generate SQL Commands

In the final step, the tool produces the necessary SQL commands to establish role-based access control within a relational database. The identified roles are created along with the necessary grants and revokes to established the appropriate privileges.

For the sentence, "A nurse can order a lab procedure for a patient," the following commands would be generated:

```
CREATE ROLE nurse;
GRANT SELECT, INSERT on lab_procedure TO nurse;
GRANT SELECT, INSERT on patient TO nurse;
```

Additionally, the tool generates a report with access conflicts or other validation issues. The process compares each IR to all of the other IRs to find conflicts which occur when one sentence gives a role access to a table, but another sentence would revoke the same role's access from the table. The report also includes any subjects or actions not mapped to their corresponding database elements as well as any sentences classified as access control, but missing part of the access control tuple. The tool also generates a report showing the traceability of access control rules back to the originating sentences in the natural language text.

5. EVALUATION

This section details our evaluation of the process.

5.1 Application: iTrust

To evaluate the procedure, we used iTrust[8] [22] 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 2012. The application follows a typical three-tiered architecture with

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

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

logical layers for the presentation, application, and persistence. Instructors, teaching assistants, and students have contributed to the application, which is currently in its 14th version. Each class performs software maintenance on the application, correcting defects and implementing new functionality. The requirements consist of 40 use cases plus additional non-functional requirements, constraints, and a glossary. The version we used contained 1114 sentences with 409 (36.7%) of those sentences classified as access control. The application database contains 41 tables. While the application contains ten roles, all database accesses are performed with a common system account.

5.2 Process Observations

The first author spent 15 hours to process the 1114 sentences through the tool. As the number of sentences classified grew, the time spent to classify each sentence was reduced due to additional elements in the domain dictionary and more accurate classifications produced by the *k*-NN classifier. For future development iterations on the same project, we expect the time spent within the tool to be significantly less because much of the domain dictionary is already accumulated. We also expect other projects to spend less time provided they can utilize an existing *k*-NN classifier or domain dictionary.

The *k*-NN classifier performed well on sentences where there were common patterns. For instance, requirements are often phrased starting with "The system shall provide the ability to…" The produced IRs for those type of sentences all had the vertices for "system,", "provide," and "ability" in the same location and the distance metric showed zero difference for these vertices. Another common pattern handled well was in the cases were the phrase followed the pattern "<role> (desires | wishes | wants | selects) to <perform action>." The generated IRs had the domain subject flag and the domain action flag in the same location and as such the calculated distances were relatively low even with different words as "desire" or "selects" in between the two locations.

5.3 Classifier Evaluation

To evaluate our classifier, we compared precision, recall, and f-Measure to six other models utilizing a stratified *n*-fold cross-validation. With this method, data is randomly partitioned into *n* folds based upon each fold of approximately equal size and equal response classification. For each fold, the models are trained on the remaining folds and then the contents of the fold are used to test the model. The *n* results are then averaged to produce a single result. We follow Han et al.'s recommendation [12] 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 each sentence is tested just once.

For the models, we used four machine classification models across different techniques from Weka [11]. Additionally, we used two random methods, weighted and 50%. The weighted random model assumes that the classifier knows the classification of each item in the training set and randomly returns answers proportionally to the existing classifications. The 50% model has no knowledge of the training set and randomly returns answers with equal likelihood between the two classifications values. Comparison to 50% model would be appropriate when a pre-defined classifier is used by a project team. To generate input data for the Weka models, we flattened the IR graph into a comma separated list format and then executed a Weka filter to convert all string values to nominal values. Table 1 lists the results.

Overall, our model performed relatively equal to the existing Weka classifiers while substantially out-performing the random models. J48, a decision tree classifier, performed the best. However, J48 had a lower recall, which implies that a number of access control statements were not identified. While our classifier overcomes a significant limitation in Weka's classifiers in that new strings and instances can incrementally be added, the performance of existing Weka models highlights the need to reconsider whether it is appropriate for us to continue work on the *k*-NN classifier.

Table 1. Test Results of Stratified 10-fold Cross Validation

Model	Precision	Recall	F_1Measure
Weighted Random	.38	.40	.39
50% Random	.36	.50	.42
Naïve Bayes	.85	.94	.89
J48	.98	.84	.91
Ridor	.97	.84	.90
IB1	.84	.81	.83
REDE *k*-NN (k=1)	.78	.92	.85

For our classifier, we evaluated different values of k to use when finding the k-nearest neighbors. The highest F_1 Measure occurs at $k = 3$. However, 40% more false positives where generated than the next best performing value at $k = 1$. Consequently, we set k=1 to minimize incorrect access control.

We also evaluated whether or not it was appropriate for the classifier to even return a value. As we processed the iTrust requirements, we noticed circumstances in which the classifier returned results, but the nearest sentences were significantly different from the sentence being classified. We adopted a threshold value to determine whether or not the classification results should be utilized. For example, if a sentence had 10 vertices and the computed distance to its k-nearest neighbors was 7.6, we would only accept the classifier's answer if the threshold was set at 0.76 or higher. As can be seen in Table 2, low threshold values resulted in significantly higher values for precision and recall with the

disadvantage that a substantial number of sentences were not automatically classified. We adopted 0.80 as our value.

Table 2. Threshold Response Results

Threshold	% Answered	Precision	Recall	F_1 Measure
0.25	33%	0.97	0.99	0.98
0.50	56%	0.95	0.98	0.96
0.55	58%	0.95	0.98	0.96
0.60	64%	0.94	0.96	0.95
0.65	69%	0.92	0.97	0.94
0.70	75%	0.88	0.94	0.91
0.75	85%	0.84	0.95	0.89
0.80	92%	0.83	0.95	0.89
0.85	96%	0.80	0.94	0.86
0.90	98%	0.73	0.93	0.82
0.95	99.8%	0.78	0.93	0.85
1.00	100%	0.78	0.92	0.85

5.4 Database Table Coverage

Step 3 of REDE involves mapping resources found in the natural language text to physical database tables and columns. In our evaluation, we successfully mapped to 92% of the 41 tables. A table for password failures was missed as the related use case documented the scenario across multiple sentences and our resolution method did not handle the condition properly. For the "report request" table we assumed that this functionality expressed in the use cases did not require persistence. Finally, there was a table to store global variables that was not mentioned within the use cases. We identified several objects in the use cases that did not mapped to the database. Most of these were involved specific user roles and their information was stored in the "users" table. The other issue involved "immunizations" and this information was contained with application code. Overall, the process was effective at mapping tables, but missed situations where tables references were not present in the natural language text or were implementation dependent.

5.5 System Execution

From the generated SQL commands, we evaluated iTrust with the database access control in place. As iTrust uses a common system account to access the database, accounts were established for the different users based upon their roles in the database. The application was modified slightly for the user's database authentication credentials to serve as the authentication for the application and to establish the database access for that user's session. Production deployments will need to consider the impacts on connection pooling. We then executed the 28 existing acceptance tests for iTrust to ensure the system performed correctly. As expected from the results in the previous section, we had to grant additional permissions for tables not granted any access. Additionally, there were other issues found with functionality not fully described in the use cases such as logging that were required additional privileges to be granted. Once all of the additional privileges were added, all 28 acceptance tests passed. We also defined and successfully executed five system tests to show access control was enforced appropriately at the database level in the event the application level access control was bypassed.

6. LIMITATIONS

This section presents the limitations of this work.

6.1 Process

Within the REDE process and tool, we assumed only one access control tuple would exist within a sentence. However due to conditions or clauses within a sentence multiple tuples may exist. This problem can be solved through identifying parts of the IR that related to a specific clause or condition.

Another limitation encountered that our lookup into the domain dictionary did not take into account whether or not a noun acted as sentence subject as an object. As such, subject and resource domain flags were set for certain words. The user then needed to manually correct the flags. To fix this limitation, the subject lookup would only be applied to vertices (words) that had an edge that specified the relationship was the nominative subject or the agent in a passive sentence.

The process did not utilize the document's structure (i.e., when a use case started or the different sections within a use case) as it parsed paragraphs at a time. Utilizing the structure to identify the start locations of test scripts, use cases, and other items would allow a tool to infer greater information about the current text under analysis.

Within the domain dictionary, we only stored one entry for each action verb discovered in the document. We discovered situations in which a verb such as "order" or "request" had very different implications for access depending upon its semantic meaning. "Request" could imply that a user is just asking for data, which implies select permission for the database. In other cases, "request" could imply that a user wants a specific activity to be accomplished, which would imply insert permission.

For applications to effective use this process, they must stop utilizing a common system account to access the database and setup individual users in the correct roles in the database. Companies may utilize identity management software to facilitate this user establishment.

6.2 Evaluation

In evaluating the process and the tool, we only examined 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.

As one of the authors classified all of the sentences, an external threat to validity exists as to if the classifications

are correct. To check if this was a significant issue or not, we had four software developers with an average work experience of nine years classify a representative sample of ten sentences to determine whether or not those sentences implied access control. Utilizing R[9], we computed Fleiss' fixed marginal, multi-rater kappa statistic as .84 which indicates significant agreement among the five raters.

7. FUTURE WORK

We plan to evaluate our process with other applications to evaluate the process as well as utilizing a trained classifier and existing domain dictionary. From there, we plan to evaluate the process across applications domains. Ideally, we will demonstrate that our process is not limited to the healthcare domain and significant time savings can be achieved through the use of existing dictionaries and classifiers.

Through our examination of the iTrust requirements, we noticed significant number of places where individuals had access to a table, but their access was limited to specific records based upon specific conditions. To restrict access to those records, we plan to extend the process to generate database views with those conditions in place. Roles would then be granted access to those views rather than the underlying table(s). Additionally, we plan to include conditions and obligations based upon various grammatical constructs in the natural language text.

The REDE process could also be extended to validate the work against existing RBAC models for the application layer as well as exploring the use of different access control models besides RBAC.

8. CONCLUSION

In this paper, we present a new process, REDE, and tool that assist developers and security experts in automatically generating RBAC statements to send to a DBMS. We have shown how the tool detects conflicts, performs traceability from source sentences to access control rules, and visualizes coverage of existing database tables versus those specified in documentation. To best of our knowledge, we produced the first methodology to automatically classify sentences as access control related or not. We evaluated the process and tool on an existing system by processing 1114 sentences. Our *k*-nearest neighbor classifier with a unique distance metric had a precision of 0.78 and a recall of 0.92, outperforming the random guess, which had a precision of 0.38 and a recall of 0.40. The process correctly identified and mapped 92% of the physical database tables to the resources found in the requirements specification. We demonstrated the ability for the application to successfully execute with both application and persistence layer role based access control in place.

For the practitioner, we have developed a tool to incorporate into any software engineering process to detect conflicts in access control statements specified in natural language text and to save costs while establishing and maintaining RBAC at the persistence layer.

9. ACKNOWLEDGMENTS

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.

10. REFERENCES

[1] 2011 CWE/SANS Top 25 Most Dangerous Software Errors, http://cwe.mitre.org/top25/. Accessed: 2011-11-14, 2011.

[2] Androutsopoulos, I., Ritchie, G.D. and Thanisch, P., Natural Language Interfaces to Databases - An Introduction. Journal of Natural Language Engineering. 1, (Sep. 1995), pp29-81, 1995.

[3] Batini, C., Lenzerini, M. and Navathe, S.B., A Comparative Analysis of Methodologies for Database Schema Integration. ACM Computing Surveys. 18, 4 (Dec. 1986), pp 323-364, 1986.

[4] Brodie, C. a, Karat, C.-M. and Karat, J., 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, 2006.

[5] Chen, P.P.-S., English Sentence Structure and ER Diagrams. Information Sciences. 29 (1983), pp 127-149, 1983.

[6] Choi, N., Song, I.Y. and Han, H., A Survey on Ontology Mapping. ACM Sigmod Record. 35, 3 (2006), pp 34-41, 2006.

[7] Contract Worker Steals Personal Data On 9 Million Israelis, http://www.darkreading.com/database-security/167901020/security/privacy/231901478/contract-worker-steals-personal-data-on-9-million-israelis.html. Accessed: 2011-10-30, 2011.

[8] Dixon, R.M.W., A Semantic Approach to English Grammar. Oxford University Press, USA, 2005.

[9] Doan, A.H. and Halevy, A.Y., Semantic Integration Research in the Database Community : A Brief Survey. AI Magazine. vol. 26, no. 1, 2005.

[10] Donston-miller, D., Ensuring Secure Database Access., 2011.

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

[9] http://www.r-project.org/

[12] Han, J., Kamber, M. and Pei, J., Data Mining: Concepts and Techniques. Morgan Kaufmann, 2011.

[13] Hartmann, S. and Link, S., English sentence structures and EER modeling. Proceedings of the fourth Asia-Pacific conference on Comceptual modelling-Volume 67 (Ballarat, Australia, 2007), pp 27–3, 2007.

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

[15] Huddleston, R. and Pullman, G., The Cambridge Grammar of the English Language. Cambridge University Press, 2002.

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

[17] Jurafsky, D. and Martin, J., Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Pearson, 2009.

[18] Kalfoglou, Y. and Schorlemmer, M., Ontology Mapping: the State of the Art, The Knowledge Engineering Review. vol. 18, no. 1 (Jan. 2003), pp 1-31, 2003.

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

[20] Manning, C., Raghavan, P. and Schütze, H. 2008. Introduction to Information Retrieval. Cambridge University Press.

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

[22] Meneely, A., Smith, B. and Williams, L., Appendix "iTrust Electronic Health Care System: A Case Study". Software and Systems Traceability, Springer, London 2012.

[23] Omar, N., Hassan, R. and Arshad, H., Automation of Database Design Through Semantic Analysis. Proceedings of the 7th WSEAS International Conference on Computational Intelligence, Man-machine Systems and Cybernetics (Cairo, Egypt), pp 71-76, 2008.

[24] Po, L. and Sorrentino, S., Automatic generation of Probabilistic Relationships for Improving Schema matching. Information Systems. vol. 36, no. 2 (Apr. 2011), pp 192-208, 2011.

[25] Popescu, A., Armanasu, A., Etzioni, O., Ko, D. and Yates, A., Modern Natural Language Interfaces to Databases: Composing Statistical Parsing with Semantic Tractability. Proceedings of the 20th international conference on Computational Linguistics (Geneva, Switzerland), pp 141-148, 2004.

[26] Rahm, E. and Bernstein, P., A survey of approaches to automatic schema matching. The VLDB Journal. vol. 10, no. 4 (Dec. 2001), 334-350, 2001.

[27] Samarati, P. and di Vimercati, S, Access Control: Policies , Models , and Mechanisims. Foundations of Security Analysis and Design. Springer Berlin / Heidelberg. 137–196, 2001.

[28] Schwitter, R., Controlled Natural Languages for Knowledge Representation. Proceedings of the 23rd International Conference on Computational Linguistics (Beijing, China), 1113–1121, 2010.

[29] Shi, L. and Chadwick, D., A Controlled Natural Language Interface for Authoring Access Control Policies. Proceedings of the 2011 ACM Symposium on Applied Computing (TaiChung, Taiwan), pp 1524-1530, 2011.

[30] Xiao, X., Paradkar, A. and Xie, T., Automated Extraction and Validation of Security Policies from Natural-Language Documents, 2011

[31] ANSI/ISO/IEC 9075-2:1999, Information technology - Database languages - SQL - Part 2: Foundation (SQL/Foundation). ISO, 1999.