# Capstone Proposal: Converting an English Sentence to a Format Understandable by Computers

Tyler Baylson, Andrew Sexton

Spring, 2020

## 1 Introduction

A benefit of file systems, whether locally housed or "in the cloud", is the increased work efficiency from users sharing the same resources. But given the growing desire for companies to give their employees this level of efficient access to data, that data is at risk anytime security isn't easy to establish for its users. And, in the worst cases, where something like a credential file may itself be accessed by a user who should not have access, a security failure cascade may be initiated that could lead to very serious problems for the company and its users.

Given this, it is extremely important that users can easily and efficiently control access to their data, starting from simple and straightforward privacy purposes all the way to fully customized schedules of access to highly confidential data.

In order to meet this goal, easy and effective means to protect information and resources must be made available to those responsible for its security. Being that natural language is the most natural way to input a command, we propose a program which allows for natural input that will both implement policies to build secure access and receive inquiries to grant or deny access to the inquirer.

Our system will use a well-formed notation (described below) for both specifying access control policies and authorizing access when inquiries against data are made. In this way, the definition of policies becomes simpler and more efficient, which will allow users to protect their assets under their own terms, and will ensure that requesters can feed access requests to the system just as simply.

# 2 Problem Definition

All types of data in a file system are at risk when access is not defined or if that access is defined in an unclear or incomplete manner. Our system will take user input and determine the intent of a given security policy or access inquiry for the extant data on a network or filesystem which implements our access control methods.

While typical access for a given file may be legitimate, when the owner of data is not able to (or aware of how to) adjust or monitor access (especially given the complexity of certain team-based file access), it's easy for security requirements to relax. The data goes either unsecured or unmonitored, increasing unnecessary risk.

A given network with subjects, objects, and defined relationships can be interpreted as a directed labeled graph (vertices connected by edges)

1. $N$ is a finite set of subjects ($S$) and objects ($O$), that may include an environmental condition.
2. $\Sigma$ is a finite set of relationship types (labels for connections).
3. $W$ is a number attached to a relationship between two subjects. This is a percentage of trust between two users.
4. $E \subseteq N \times N \times \Sigma \times W$ represents the edges of existing connections, the relationships between subjects and objects, objects and other objects, and represents a general overview of the possible authorized interactions within that network[1].

Though programs can be written for security management, they would simply be another step in the chain that would require attention, setup, and maintenance. Much research has been performed, some of which is cited within this proposal, but no specific approach has proven quite flexible enough for general use within social networks.

Our system must process natural user command sentences in a given structure within a basic UI, resulting in appending well-formatted data to a policy file in the user's directory. This will require a simple UI for access control input, which could easily be presented within an intranet-based web-application or desktop application. It will be flexible and responsive, as companies may wish to have more granular control over its presentation. In the event of an error in input or authorization attempt, depending on certain circumstances, the program may ask for further clarification. A program already exists which will manage reading the policy rules and acting on user queries, and it also handles conflicts, giving priority to most recent policy input.

We consider that this program will be useful for companies that emphasize team-based access to shared files, especially where leadership is comprised of users for whom it is not time-efficient to learn the details of things like 'chmod' commands. Even Western Carolina

University would benefit from implementing this approach, and it goes without saying that more general social networks such as Facebook might benefit from users who would like to take a more involved stance on how their posted files are seen by other users.

## 3 Solution

Numerous solutions to access control have been theorized: Fong et al. defined an access control model that employed authorization decisions in terms of the relationships between the resource owner and a resource requester [2]. This solution, however, only addresses the case of simple expressions of policies and become equally simple rules – this does not reflect the complex nature of actual networks as demonstrated above. Aktoudianakis et al. defined a syntax for specifying a social network and its associated access control policies, which allows for rather simple generation of policies. But the syntax discussed does not implement the direct association with users to define effective policies [3]. Some solutions have suggested an XML-based format, including Extensible Access Control Markup Language (XACML). This language uses XML to generate both an access control policy language and a request/response language, but a simpler approach would be able to process policies and inquiries in natural expressions without the constraints of XML tags [4].

Our solution stands as a faithful implementation of Dr. Morovat's own Policy Based Language for Access Control [1]. It will simultaneously respect the complexities of a network with subjects, objects, and environmental availability conditions while also achieving a simpler result. The Policy-Based Attribute Access Control model will allow its users to use a natural language to communicate policies, empowering authorized users to take control of their data and reduces risk using natural language requests such as "Bob can edit my tasklist documents on weekdays" and "only I can edit my tasklist documents after 1pm Friday".

Users will define policies and make inquiries in English via an access control interface. The program will transform user defined policies, using the well-formed format described below, into system rules. It will also transform user inquiries into the well-formed format and process these as system queries.

Given sentences will be in the format: Subject verb object [conditional(time/date)]. As an example, "Alice is allowed to access photos." The system will recognize 'Access' words like "Read", "write", and "modify", including common negations. Objects themselves may be referenced by filetype (i.e. "photos" = jpg, gif, etc, and "documents" = txt, docx, etc.). It will differentiate between types of entities in a given network as modeled above:

- Target users, which get policies protecting them from unauthorized access
- Target resources, which get policies protecting them from unmanaged actions

It will group inquiries into two groups:

- Attempting to perform an operation against resources like updating or deleting resource
- Attempting to access target users (e.g. Facebook may consider this in terms of 'tagging' in a photo or 'poking' a target user)

The well-formed format template we will use indicates the following:

- Words in policy/ inquiry appear with order: subject, verb, object
- Words in rule/request appear with order: subject, action, object

The system itself will operate on, at the very least, a simple UI which may be used as a framework for accessing resources in various file system configurations and social networks. Please refer to Table 1 for our MoSCoW table of further feature considerations.

Users will start our program via command line or graphical application. They may then type access control commands into the program. These commands may be written as natural-sounding sentences. The program will then parse the sentence into individual words. These individual words will be used with WordNet's lexical database to ascertain their grammatical and syntactical significance. The program will use the grammatical and syntactical data to generate a well-formed policy rule for access control based on the original natural-sounding sentence. The newly generated policy will be cross examined with an existing policy text file to ensure the same policy does not already exist. If the newly generated policy does not exist in the policy text file, it will be appended to the policy text file. Dr. Morovat's own program handles the security activities as determined by the policy rule or inquiry.

# 4 Plan

Please refer to Table 2 for our semester schedule of implementations.

We will utilize the pytest framework to help streamline the development process and encourage test driven development for the system as it gains complexity. On that note, we have also reviewed how specific courses we have completed will reflect in this project:

1. CS 150 & 151, Introduction to Problem-Solving I & II: Basic python comprehension and working with data structures
2. CS 263, Software Engineering: Breaking down the project into tasks, preparing proposal, specifics like the MoSCoW analysis, etc.
3. CS 351, Data Structures and Algorithms: Graph data structures and traversal

4. CS 352, Programming Languages: Further functional and object-oriented data processing, preparaing input for parsing

5. Math 270, Statistics: Any analytics we perform in Capstone II are likely to build on the principles of 270

6. Math 310, Discrete Structures: Learning about graph structures and, more specifically, markov chains, which are applicable to our WordNet approach

7. And on our own, we will be learning learn PyTest and approaches to implement a theoretical model in programming practice (PBAAC).

# References

[1] Katanosh Morovat and Brajendra Panda. Policy language for access control in social network cloud. *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, pages 313–318, 2016.

[2] Philip Fong and Ida Siahaan. Relationship-based access control policies and their policy languages. In *SACMAT '11: Proceedings of the 16th ACM Symposium on Access Control Models and Technologies*, pages 51–60, New York, NY, USA, 01 2011. Association for Computing Machinery.

[3] Evangelos Aktoudianakis, Jason Crampton, Steve A. Schneider, Helen Treharne, and Adrian Waller. Policy templates for relationship-based access control. *2013 Eleventh Annual Conference on Privacy, Security and Trust*, pages 221–228, 2013.

[4] Ernesto Damiani, Sabrina De Capitani di Vimercati, Stefano Paraboschi, and Pierangela Samarati. A fine-grained access control system for xml documents. 5(2):169–202, 2002.

| Must Have | Should Have | Could Have | Won't Have |
|---|---|---|---|
| PBAAC Implementation | Basic GUI with function calls for business embedding and customization | Unstructured user sentences | Language support other than English |
| Appending distinct well-formed policies to a text file | Recognition of grammatical differences | Spellchecker | |
| | | Auto-complete | |
| | | Encryption of Policy File | |
| | | Machine learning applied to analytics | |
| | | Social Network Demo | |

Table 1: MoSCoW Board

| Date | Deliverable | Item Due |
|---|---|---|
| 2/13 | Proposal with Timetable | |
| 2/27 | Input via interface, accept and validate, handle errors, test on frameworks | |
| 3/12 | Sorting words into parts of speech, recognize access actions ("read" / "write" / "access") and their negations, poster content preparation | |
| 3/26 | Accept specific object references ("house.png"), Accept generalized object references and groups by filetype ("photos", "documents") | |
| 4/09 | GUI implementation / function calls, Generate well-formed format, Append the formatted rule to user policy file, Test with policy reader program | |
| 4/12 | | Poster due |
| 4/19 | | Poster demonstration |
| 4/23 | Confirm full functionality with policy reader program, Capstone I scope complete | |

Table 2: Project Time Table