CS 1699: Privacy in the Electronic Society

Project 2

Released: Friday, March 23

Due: Friday, April 6, 11:59 PM

Motivation

In this course, we discussed several approaches to representing and enforcing access control policies. In this project, you will implement and test a prototype enforcement engine for an access control policy language of your choice.

Your submission should consist of the following components.

- A writeup that completes the W tasks below and discusses your approach to the C tasks.
- Code that satisfies the C tasks, in the language of your choice (check with Mingda if you're not sure it will work for him)

All features, bugs, and other details with your code should be made clear in your writeup (i.e., do not submit a separate README or expect your TA to read every comment in your code). Each writing task should be clearly titled, and each code task should be clearly discussed in the writeup. In short, do not make your TA search for the components of your submission. Show off the hard work you did!

Tasks

Task W0: Choose an access control language that supports indirection, delegation, and at least 2 other features. You may consider forms of ABAC, ReBAC, SD3, or RT, but you are **not** limited to these options! We would be happy for you to choose something unique and impress us!

Your language must support at least the following features:

- *Indirection*: The ability to assign a large set of permissions at once. Consider roles, attributes, groups, etc.
- *Delegation*: The ability to state that an access should be allowed if a separate (named) entity declares a certain fact. Consider RT's delegation of attribute authority, or the $ notation in SD3.
- *At least two others*: At least two other features that allow administrators to write expressive policies. Consider role inheritance (role hierarchy), attribute intersection, parameterized attributes, etc.

You may include more than the required features for extra credit.

To start your writeup, overview the access control system that you chose, including what features it supports.

Task W1: Fully describe the syntax used to write a policy file for your access control language. This file should be plaintext and simple for an administrator to write. Explain how each supported feature is represented in plaintext. If your file needs to include a list of all existing users/subjects or files/resources/objects, explain the format for this information as well.

You may consider standard formats for structured input, which would enable you to utilize existing parsing libraries. Consider JSON, XML, datalog, etc. Note that you are not required to use the same syntax that was presented in lecture (for instance, you may consider an XML encoding of SD3 rather than a datalog encoding).

Task C2: Implement a program to interpret your access control language. It should be able to do the following steps:

1. Read, parse, and preprocess (if necessary) a specified policy file in the format you describe in Task W1.
2. Prompt the user for an access query (for example, a query of the form, "Can user *u* access file *f* with *read* privilege?"). Clearly state the format in which the user must type their query.
3. Using the policy that was parsed from the input file, determine whether the access should be permitted, and output the response (*allow* or *deny*).
4. Repeat from Step 2.

Note that you are only required to support access queries, but you can implement additional queries for extra credit (e.g., "Is user *u* a member of role *r*?").

Task W3: Explain, with examples, how to implement/encode *indirection* in your language. You should give at least 2 examples to make it clear how this feature works.

Task W4: Explain, with examples, how to implement/encode *delegation* in your language. Again, you should give at least 2 examples.

Task W5: Name the 2 other access control features that you implemented in your program. Explain, with examples, how to implement/encode these features in your language. As before, give at least 2 examples of each feature.

Task C6: Construct at least 4 policies (manually written or randomly generated) of widely varying size/complexity. Your goal is to study the efficiency of enforcing policies of various complexities to investigate how your program scales to large, complicated policies. Consider, for instance, long chains of trust.

Write a program to time (benchmark) your processing of a policy and query. This program should not be interactive and should run hardcoded (or randomly generated) queries on the policies you constructed.

You should measure separately the time two distinct tasks for each policy:

1. The time it takes your program to read and preprocess the policy, ignoring queries.
2. The time it takes your program to respond to a query after the policy has been read and processed.

Thus, you should have at least 2 measurements for each experimental policy. Use the average of multiple trials if your timing is not consistent.

Task W7: Interpret the results of Task C6. Show a chart, table, or graph of your data. Explain what this tells you about how your enforcement engine scales. If the runtime scales poorly, is this a problem with the language you chose, or with your specific enforcement engine? Is this language reasonable to use for personal computers? What about enterprise scenarios? Can it scale to a large cloud storage service? Show off as much as you can regarding what this experiment taught you about access control enforcement.

Problem / Algorithm Choice

Up to 10 bonus points will be awarded for choosing an interesting access control language and incorporating more than the required features. More points will be awarded for selections that more thoughtful, interesting, creative, clever, etc.

Grading

Task	Points
Task W0	10 (bonus)
Task W1	20
Task C2	35
Task W3	5
Task W4	5
Task W5	10
Task C6	10
Task W7	15
Total	**Up to 110**

Note

As this course is an upper-level elective, you are being given a lot of freedom in terms of how you tackle this project. In exchange, you also have a lot of responsibility to demonstrate your hard work adequately to the TA. As such, there are tasks in this assignment that require you to discuss your code in detail. Your discussion should closely align to, and refer to, your specific implementation. You may use any programming language and library functions that you prefer, but you must ensure that your choice of language/library does not “do the work for you.” For instance, you may use parsing libraries to interpret the text policy, but you should not use any packages that determine access control decisions or automatically interpret datalog. This is a 2-week project in a senior-level course; if you satisfy the requirements in only 4 lines of code, you’re probably using a library function that allows you to skip the hardest parts of the assignment. If you need help deciding whether something should be permitted, contact your instructor and/or TA.

Submission

Upload your code and writeup to the Box folder created for you with the name `cs1699-p2-abc123`, where `abc123` is your Pitt username.