Arizona State University
School of Computing, Informatics and Decision Systems Engineering

CSE 365 – Spring 2019

Assignment 1

DUE: Tuesday, February 26, 2019, 3:00 PM

SUBMISSION INSTRUCTIONS: We will provide an submission link on Blackboard for you to submit all of your source code (no executables), as well as a `make` file that will compile, prepare your code for execution. A sample `make` file for you to customize is now available in the *Assignment 1* folder in our class Blackboard site. Detailed submission instructions will follow shortly on our class Blackboard site. **Do not wait until the submission link is up to start working on your assignment!**

You are expected to implement your code from scratch, without resorting to any third-party libraries or APIs. All submissions will be checked for plagiarism, and all detected cases will be promptly reported to ASU authorities. Please direct any questions or clarifications regarding this assignment to the discussion forum labeled as *Assignment-related Questions* within the class Blackboard site.

1. **Implementing an ABAC Access Control Engine (100 Points).**

   For this assignment, you will implement an access control engine depicting the ABAC model as it was discussed in class. Your engine should be able to load a **syntactically-correct** ABAC policy, that is, it should be able to parse the policy from a text file, store the policy within some internal data structures, and been able to respond to an authorization request asking if a given user can be granted or denied a given permission.
   In addition, your engine should allow for modifying the loaded policy by adding or removing permissions, entities, e.g., users and objects, and by changing the *attribute-assignment* (AA) and the *permission-assignment* (PA) relations as discussed in class.

   You are free to implement your ABAC engine using the programming language of your choice. However, **you are required to implement the command line interface** we discuss next, so we can properly test your engine and we can correctly award you all the points you may deserve for your hard work. Please be advised that any deviations from the expected behavior of your code, e.g., it fails to compile or implement the requested command line interface correctly, will result in valuable points being deducted from your final assignment grade.

Overall, the following commands must be supported:

`load-policy` *policy-file-name*

Parses a text file identified by *policy-file-name*, containing an ABAC policy in the format shown in the class slides, and loads it into the internal data structures implemented by your engine. You should be able to load any policy that implements such a format, and **you can expect all policies to be syntactically correct**, that is, there is no need to implement an advanced parser that can detect and react to formatting issues within *policy-file-name*. If a given policy has syntax errors, you can just abort execution gracefully. In case the command succeeds or fails, **no message must be shown to the screen**.

`show-policy`

Displays an already-loaded policy into the screen. For simplicity, you can use the same format as shown in class, or implement your own. If no policy has been loaded beforehand, no output will be produced by your engine. Since this is just an auxiliary command, e.g., it will not be tested for grading purposes, there is no need for the format displayed by this command to match exactly the one contained in a text file containing the ABAC policy that was loaded beforehand.

`check-permission` *user-name object-name environment-name permission-name*

Checks if the permission identified by *permission-name* over the resource identified by *object-name*, can be granted to the user identified by *user-name* under the execution environment labeled as *environment-name.* If the permission can be granted, a message of the form: `Permission Granted!` should be outputted to the screen. Otherwise, the message `Permission Denied!` should be shown. Both messages should terminate with a carriage return (`\n`). **Do not include any extra information in your response message**.

`add-entity` *entity-name*

Adds an entity identified by *entity-name* to a previously-loaded policy. In case the command succeeds, or in case the command fails, e.g., no policy has been loaded, or *entity-name* is already defined in the policy, **no message must be shown to the screen**.

`remove-entity` *entity-name*

Removes an entity identified by *entity-name* to a previously-loaded policy. In case the command succeeds, or in case the command fails, e.g., no policy has been

loaded, or *entity-name* was not listed in the policy, **no message must be shown to the screen**.

`add-attribute` *attribute-name attribute-type*

Adds an attribute identified by *attribute-name* of datatype *attribute-type* to a previously-loaded policy. In case the command succeeds, or in case the command fails, e.g., no policy has been loaded, or *attribute-name* is already defined in the policy, **no message must be shown to the screen**.

`remove-attribute` *attribute-name*

Removes an attribute identified by *attribute-name* to a previously-loaded policy. Also, it removes the attribute identified by *attribute-name* from all PA assignment entries involving it. If no other attributes were also present in the assignment, the PA assignment entry is removed completely. In case the command succeeds, or in case the command fails, e.g., no policy has been loaded, or *attribute-name* was not defined in the policy, **no message must be shown to the screen**.

`add-permission` *permission-name*

Adds a permission identified by *permission-name* to a previously-loaded policy. In case the command succeeds, or in case the command fails, e.g., no policy has been loaded, or *permission-name* is already defined in the policy, **no message must be shown to the screen**.

`remove-permission` *permission-name*

Removes a permission identified by *permission-name* to a previously-loaded policy. Also, it completely removes all PA assignment entries involving *permission-name.* As with all previous commands, the command should return quietly, that is, **no message must be outputted to the screen** if it succeeds or fails.

`add-attributes-to-permission` *permission-name* [*attribute-name attribute-value*]

Assigns the permission identified by *permission-name* to a non-empty list of pairs consisting of an attribute identified by *attribute-name* and a value of *attribute-value*, thus modifying the PA relation consequently. **This command creates brand new entries in the PA relation, e.g., it does not try to modify existing ones**. Also, it assumes *permission-name* and all *attribute-names* contained in the list of pairs exist in the ABAC policy, and also assumes that, for each *attributename* and *attributevalue* pair, *attributevalue* corresponds to the *attribute-type* defined for *attributename* in the policy. If any of these do not exist in the policy, or in case one of the attribute pairs is incomplete, e.g., either it is missing the *attribute-name* or

attribute-value component, the command fails. As with all previous commands, the command should return quietly, despite of the result, that is, **no message must be outputted to the screen** if it succeeds or fails.

`remove-attribute-from-permission` *permission-name attribute-name attribute-value*

Removes the assignment of the permission identified by *permission-name* to the attribute identified by *attribute-name*, thus modifying the PA relation consequently. In case the command succeeds, or in case it fails, e.g., *permission-name* and *attribute-name* were not previously related, or one of the two is missing in the policy, **no message must be shown to the screen**.

`add-attribute-to-entity` *entity-name attribute-name attribute-value*

Assigns the attribute identified by *attribute-name*, with value *attribute-value* to the entity identified by *entity-name*, thus modifying the AA relation consequently. This command assumes *attribute-name and entity-name* have been previously defined in the loaded ABAC policy. In addition, it assumes *attribute-value* corresponds to the datatype defined for *attribute-name* in the loaded policy. As with all previous commands, the command should return quietly, that is, **no message must be outputted to the screen** if it succeeds or fails.

`remove-attribute-from-entity` *entity-name attribute-name attribute-value*

Removes the assignment of the attribute identified by *attribute-name* with value *attribute-value* to the entity identified by *entity-name*, thus modifying the AA relation consequently. Also, this command assumes both *entity-name* and *attribute-name* do exist within the currently-loaded ABAC policy, failing gracefully if that is not the case. As with all previous commands, the command should return quietly, that is, **no message must be outputted to the screen** if it succeeds or fails.

Sample Command Line Execution

The aforementioned commands will be executed in sequence, separated by a semi-colon, producing a single output to the screen as a result of invoking the `check-permission` command at the end. In the following example, a policy called `Example-ASU.txt`, which is shown below and is similar to the one listed in the class slides, is first loaded, and a permission on a user called “Josie” is checked next. Once a response message is shown to the screen, your engine should terminate quietly. You are expected to parse the sequence of commands, separate each command, and execute it subsequently. Also, you should make sure your ABAC Engine can be called by the `abacmonitor` name in

the command line, by properly customizing the sample `make` file that will be provided in the class Blackboard site.

Sample ABAC Policy Example-ASU.txt

```
ATTRS = <Role, role>; <String, fileName>; <String,
fileOwnerName>;<AbstractTime, currentTime>
PERMS = <PW>; <PR>
PA = <fileName, "grades.txt">; <role, "Student">;
<currentTime,WithinOH> : PR - <fileName, "grades.txt">;<role,
"Professor">; <currentTime, WithinOH> : PR - <fileName, "grades.txt">;
<role,"Professor">; <currentTime, WithinOH> : PW - <fileName,
"grades.txt">;<role, "Dean">; <currentTime, WithinOH> : PW -
<fileOwnerName,"Carlos"> : PR - <fileOwnerName, "Carlos"> : PW
ENTITIES = <Josie>;<Carlos>;<Kyle>;<ENV>;<GradesFile>;
AA = <Josie> : <role, "Student">; <Carlos> : <role,
"Professor">;<Kyle> : <role, "Dean">; <ENV> : <currentTime, WithinOH>;
<GradesFile>: <fileName, "grades.txt">;
```

Sample Commands

```
> abacmonitor "load-policy Example-ASU.txt; check-permission
Josie GradesFile ENV PW"
Permission DENIED!
```

In another example, the sample `Example-ASU.txt` policy is first loaded, a permission is added to the <role, "*Student*"> attribute, and then the same permission is checked for the user "Josie". The result of executing the command differs from the previous example as a resulting of adding a new permission to the <role, "*Student*"> attribute.

```
> abacmonitor "load-policy Example-ASU.txt; add-attributes-to-
permission PW role \"Student\"; check-permission Josie
GradesFile ENV PW"
Permission GRANTED!
```