Access Control Library Documentation

1. Compilation

This library is written for Python3 and uses the following modules to function properly:

- Json
- Collections
- Sys
- OS

To call library functions, one simply needs to make the following call:

```
$python3 auth.py "command" parameter1 parameter2 ...
```

where command is one of AddUser, Authenticate, SetDomain, DomainInfo, SetType, TypeInfo, AddAccess, or CanAccess and parameters are the necessary user information for the corresponding function call as described in section 3 below.

2. Setup

No additional setup is required. The library automatically creates a hidden *.tables* folder if one does not exist and uses it to store persistent data in the form of JSON files. An existing *.tables* folder will not be overwritten but it is not recommended to have files within it named *acm.json, domains.json*, *types.json,* or *passwords.json* as the library creates and uses files of those names in order to maintain data persistence. The library will not function properly if it reads data from files it did not create itself (as they are not intended to be used for this library) so it is recommended to run the library calls in a new directory without a *.tables* folder at all. It is therefore best practice to let the API create these files as needed by running the library commands in an empty directory with just the auth.py file.

3. Functions

This is a description of what the API calls do, and the necessary parameters needed for them. Having too few or too many parameters will produce a corresponding error output. Successful executions will output a "Success" message.

a. **AddUser user password**
   Adds (user, password) key value pairing to passwords dictionary stored in *.tables/passwords.json* (created if it does not exist). Outputs a missing username error if username is empty string.
b. **Authenticate user password**
   Checks if (user, password) matches the key value pair corresponding to the user in *passwords.json*. Outputs a no such user error if username is not in the dictionary or if the file has not been made yet (i.e. AddUser has not been called yet).
c. **SetDomain user domain**

Adds (domain, [users]) key value pairings to a domains dictionary stored in *.tables/domains.json* (created if it does not exist). Adds the user to a list of users corresponding to the domain if it already exists, otherwise creates a new domain in the dictionary with the user as the only element in the list. This list is also checked to make sure there are no duplicates but adding a duplicate user to a domain will still return a Success. Returns a no such user error if the user does not exist yet in the system and a missing domain error if the domain is an empty string.

d. **DomainInfo domain**
Returns the list of users under a domain name stored in *.domains.json*. Will return nothing if the domain does not exist in the file and a nonexistent domain is treated the same as a domain with no users. Outputs a missing domain error if the domain is an empty string.

e. **SetType object type**
Adds (type, [objects]) key value pairings to a types dictionary stored in *.tables/types.json* (created if it does not exist). Adds the object to a list of objects corresponding to the type if it already exists, otherwise it creates a new type in the dictionary with the object as the only element of the list. This list is also checked to make sure there are no duplicates but adding a duplicate object to a type will still return a Success. Returns a missing type or object error if the type or object are empty strings.

f. **TypeInfo type**
Returns the list of objects under a type name stored in *.types.json.* Will return nothing if the type does not exist in the file and a nonexistent type is treated the same as a type with no users. Outputs a missing type error if the type is an empty string.

g. **AddAccess operation domain type**
Builds the access control matrix through a dictionary of dictionaries stored in *.tables/acm.json* (created if it does not exist). Adds the operation to the corresponding dictionary if the domain and type already exist. Otherwise, if the domain and/or type are missing, empty lists are added in the corresponding *.types.json* or *.domains.json* files for the new type and/or domain. Once they are created, the access operation will be added to the *acm.json* file. Dictionary is indexed by domain first and then type and stores a list of operations that the corresponding domain/type has access to. Even for already existing access operations, the operation will not be duplicated but still return a success. Will return a missing error if any parameter is an empty string

h. **CanAccess operation user object**
Returns Success if user can perform operation on object as determined by acm.json. For every domain in the list of domains the user is in and for every type the object is in, we check in *acm.json* for the operation indexed by domain and type. If the operation is found in this search, we output Success. Otherwise, we return an access denied error. Even if the user or object cannot be found or does not exist it will still only give an access denied error.

4. Basic Testing

The following basic tests were done with the auth.py command in an empty directory. Terminal outputs are provided as well as any extra functionality done quietly.

Command	Output
python3 auth.py	Error: Not enough arguments given (*.tables* folder created)
python3 auth.py add_username rakshay pword1	Error: invalid command
python3 auth.py Authenticate user password	Error: no such user
python3 auth.py TypeInfo type	
python3 auth.py CanAccess delete rakshay file1	Error: Access Denied
python3 auth.py AddUser "" pword1	Error: username missing
python3 auth.py AddUser Rakshay pword1	Success (*passwords.json* created)
python3 auth.py AddUser Rakshay pword2	Error: user already exists
python3 auth.py Authenticate Rakshay pword1	Success
python3 auth.py Authenticate Rakshay wrongpword	Error: incorrect password
python3 auth.py SetDomain Rakshay Admin	Success (*domains.json* created)
python3 auth.py SetDomain Paul Admin	Error: no such user
python3 auth.py DomainInfo Admin	Rakshay
python3 auth.py SetType rakshay_file files	Success (*types.json* created)
python3 auth.py SetType paul_file files	Success
python3 auth.py TypeInfo files	rakshay_file paul_file
python3 auth.py AddAccess delete Admin files	Success (*acm.json* created)
python3 auth.py AddAccess delete User text_files	Success (User, [] added to *domains.json* and text_files, [] added to *types.json*)
python3 auth.py CanAccess delete Rakshay rakshay_file	Success
python3 auth.py CanAccess delete Rakshay text_files	Error: access denied