Usability of Access Control Frameworks

Go through the briefing slides before starting.

ABAC Frameworks

Authzforce, **Casbin**, **OPA (Open Policy Agent)**, **Py-ABAC** and **Vakt** are access control frameworks that provide policy engines which read policies in a fixed specification language. Policy engines load in access control policies which are logical expressions and evaluate them using the input access request which is snapshot of your system representing a scenario.

Authzforce expresses access control policies in XACML Policy format of XML language and accepts access requests in XACML Request format of XML language. Please refer the following documentation for more information on XACML and refer this for XACML features supported by Authzforce.

Casbin expresses structure of access request, access control policy and its evaluation method in conf file. Additional policy rules are specified in csv file. Please refer the following documentation for more information on Casbin.

OPA expresses access control policies in Rego syntax and accepts access requests in JSON format. Please refer the following documentation for more information on OPA.

Py-ABAC expresses access control policies in JSON format and accepts access requests in JSON format. Please refer the following documentation for more information on Py-ABAC. Do check the README.md in py-abac_json directory.

Example Policy: Allow Upload Submission

Plain English

An authenticated user is allowed to upload a submission only if they are a course instructor is viewing a program in TEST_DRIVE mode, and the respective course, lab, and program are not in the trash.

Otherwise, if the course is published, the lab and program are visible, and none of them are in the trash, the submission is allowed within the lab's available timeframe, given it is coming from an allowed IP range for the lab. In this case, the user must be either a course student or a course instructor switched to student mode.

Implementation

You are given implementation of the example policy in all the four frameworks. Check how the logical expression or pseudocode of the poilcy mentioned in slides is specified in **policy-meta.conf, policy-rules.csv** (casbin policy specification) and **policy.*** (for policy specification of rest frameworks)

This is the directory structure of example policy implemented in all four access control frameworks. You need to maintain same directory structure for each task.

```
├── authzforce_xacml
│   ├── install.sh
│   ├── pdp.xml
│   ├── policy.xml
│   ├── request.xml
│   └── run.sh
├── casbin_csv
│   ├── enforcer.py
│   ├── __init__.py
│   ├── install.sh
│   ├── policy-meta.conf
│   ├── policy-rules.csv
│   ├── requests.json
│   └── run.sh
├── opa_rego
│   ├── install.sh
│   ├── policy.rego
│   ├── requests.json
│   └── run.sh
├── README.pdf
├── README.md
├── py-abac_json
│   ├── enforcer.py
│   ├── install.sh
│   ├── policy.json
│   ├── py_abac-0.4.1-py3-none-any.whl
│   ├── README.md
│   ├── requests.json
│   └── run.sh
```

Files under `<access-control-framework>_<specification-language|data-format>` directory are implementing below policy in said framework

```
./install.sh # Install required dependencies
./run.sh # Starts policy engine, loads policy and prints decision for
access request
```

Tasks

To complete the tasks, you have to modify both policy and request as required. In rare case, you may have to modify the enforcer files.

Framework	Policy File(s)	Access Request File(s)	Notes
OPA	policy.rego	request.json	
Casbin	policy-meta.conf, policy-rules.csv	request.json	

Framework	Policy File(s)	Access Request File(s)	Notes
Authzforce	policy.xml	request<number>.xml	pdp.xml holds policy engine config
Py-ABAC	policy.json	request.json	

> Make copy of the entire directory for each task and rename the directory to task<number>.

> Upload the zip file containing all the task directories in the feedback form mentioned last in this document.

> For every Task, please note down the start and end time.

> Cover every scenario possible by writing all access requests and make your policy logic robust.

Task 1

Update the example policy where authenticated users with role STA or JTA are allowed to upload submission for a program in TEST_DRIVE mode, provided the course, lab and corresponding program are not in trash. Deny the upload submission operation for users with role INSTRUCTOR in INSTRUCTOR Mode.

Task 2

Update the example policy where authenticated users with role STA or JTA in STUDENT mode are allowed to upload submission for a program provided the course, lab and corresponding program are visible, not in trash. Lab is accessed in published timeframe. Remove IP-based restriction for users with role - Instructor, STA or JTA in STUDENT mode.

Task 3

Implement a new policy to allow an authenticated user with role INSTRUCTOR or STA in INSTRUCTOR mode to evaluate all student submissions only after lab's published timeframe gets over. Authenticated users with role JTA or STUDENT should not be allowed. Any user in STUDENT mode should not be allowed. Make sure respective course, lab and program are not in trash.

Task 4

Implement a new policy to allow authenticated users to download their own submission if they are a course instructor testing the program, given the course, lab, and program are not in the trash.

Otherwise, if the course is published, the lab and program are visible, the lab submission deadline has already passed, and the course, lab, and program are not in the trash, then allow downloading the submission owned by the user. The user can be either a course student or an instructor switched to student mode, in this case.

Modify access request by including submission object.

Task 5

Implement a new policy to allow authenticated users to run visible test cases only if they uploaded submission for the program, if program has any test cases in visible state, and if submission is not evaluated for the visible testcases before.

The user can be an instructor testing the program, in which case the respective course, lab, and program must not be in the trash.

Otherwise, if the user is a student in the course or an instructor switched to student mode, they can run visible test cases only if the course is published, the lab and program are visible, none of them are in the trash and lab is accessed after published time.

Do not add testcase object in access request, frame like ProgramState example in the slides.

Task 6

Update the example policy to allow authenticated users with role INSTRUCTOR, STA or JTA to upload submissions on behalf of student, anytime after the lab is published.

Feedback Form

After completing the above six tasks in assigned Access Control Frameworks, please submit your feedback on using these frameworks through the Google form