

Usability of Access Control 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.

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](#).

Vakt expresses access control policies as Python vakt.Policy objects and accepts access requests as Python vakt.Inquiry objects. Please refer the README section for more information on [Vakt](#).

Example

```
├── 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
├── instructions.pdf
├── opa_rego
│   ├── install.sh
│   ├── policy.rego
│   ├── requests.json
│   └── run.sh
├── py-abac_json
│   ├── enforcer.py
│   ├── install.sh
│   └── policy.json
```

```
|   ├── py_abac-0.4.1-py3-none-any.whl
|   ├── requests.json
|   └── run.sh
└── vakt_python
    ├── additional_rules.py
    ├── enforcer.py
    ├── __init__.py
    ├── install.sh
    ├── policy.py
    ├── requests.json
    └── run.sh
```

Files under `<access-control-framework>_<specification-language>` 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
```

Example Policy implemented

An authenticated user is allowed to upload a submission only if they are a course instructor testing a program, 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.

Check how policy is specified in **policy-meta.conf**, **policy-rules.csv** (casbin policy specification) and **policy.*** (for policy specification of rest frameworks)

Tasks

To complete the tasks, you have to modify both policy and request as required. In case of OPA, `policy.rego` holds policy and `request.json` holds the access request. In case of Casbin, `policy-meta.conf`, `policy-rules.csv` hold the policy and `request.json` holds the access request.

For every Task, please note down the time taken to complete the task.

Task 1

Update the above example policy to restrict users(subjects) with role JTA in STUDENT mode to allow upload submission if course, lab and corresponding program are visible and are not in trash.

Task 2

Update the example policy for all users by removing IP restriction.

Task 3

Implement a new policy to allow an authenticated user to view course participants only if they are a participant in the course. Also, the course students can view participants only if the course is published.

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 have already submitted for the program, if there are any test cases in visible state, and it has not been run yet for that submission.

The user can be an instructor testing the program, in which case the 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, and none of them are in the trash.

Task 6

Modify the Allow Upload Submission to allow accept submissions on behalf of student

Keep the new policies alongwith the previous one.

After completing the above three tasks, please submit your feedback on using these frameworks through the Google [form](#)