Nefertari Documentation

Release

Brandicted

May 17, 2016

Contents

Source code: https://github.com/ramses-tech/nefertari-guards

Advanced ACLs for Nefertari.

CHAPTER 1

Table of Contents

1.1 Getting started

Add `nefertari-guards` to your requirements file or installing manually using:

```
$ pip install nefertari-guards
```

1.1.1 Requirements

- Python 2.7, 3.3 or 3.4
- Nefertari (or Ramses)

1.1.2 For Nefertari

1. add `config.include('nefertari_guards')` to your `main()`
2. subclass your model classes with DocumentACLMixin
3. subclass your acl classes with DatabaseACLMixin

1.1.3 For Ramses

- add `database_acls = true` to your .ini file

1.2 ACL Filtering

ACL filtering is one of the features of nefertari-guards. The main idea of ACL filtering is - when requesting collection (GET/PATCH/DELETE), corresponding permissions of each item are checked and if user doesn't have permission to access the item, it is dropped from resulting response.

1.2.1 How It Works

User is considered to be allowed to see collection item in results if:

1. Any of his principals are `Allow`ed either `all` or current request permission; AND

2. None of his principals are `Deny`'ed either `all` or current request permission.

Pemissions checked:

- collection GET: `all`, `view`
- collection PATCH: `all`, `update`
- collection DELETE: `all`, `delete`

Things to consider when working with ACL filtering:

1. ACEs that `Deny` supersede ACEs that `Allow`.
2. The items that a user gets on collection GET does not necessarily guarantee that all of those items will affected by collection PATCH or DELETE. E.g. if item allows `view` to user but doesn't allow `update`, that item will be visible to user on collection GET, but won't be affected by collection PATCH.
3. ACL filtering does not take into account (does not inherit) collection ACL.

1.2.2 Examples

Let's consider a user that performs a collection GET request ('view' permission) and has the following principal identifiers: "john", "group1", "everyone", "authenticated".

Items with the following ACLs will be visible to that user:

User 'john' is explicitly allowed to view an item (and not denied):

```
[(Allow, 'john', 'view')]
# OR
[(Allow, 'john', ALL_PERMISSIONS)]
```

User's group is explicitly allowed to view an item (and not denied):

```
[(Allow, 'group1', 'view')]
# OR
[(Allow, 'group1', ALL_PERMISSIONS)]
```

Everyone is explicitly allowed to view an item (and not denied):

```
[(Allow, Everyone, 'view')]
# OR
[(Allow, Everyone, ALL_PERMISSIONS)]
```

Authenticated is explicitly allowed to view an item (and not denied):

```
[(Allow, Authenticated, 'view')]
# OR
[(Allow, Authenticated, ALL_PERMISSIONS)]
```

User 'john' is explicitly allowed to view an item and access is denied to users of 'group2' to which user does NOT belong:

```
[
    (Allow, 'john', 'view'),
    (Deny, 'group2', 'view'),
]
```

User 'john' is explicitly allowed to view an item and access is denied to user 'john' to update:

```
[
    (Allow, 'john', 'view'),
    (Deny, 'john', 'update'),
]
```

Items with following ACLs will NOT be visible to that user:

User 'john' is explicitly denied to view an item:

```
[(Deny, 'john', 'view')]
# OR
[(Deny, 'john', ALL_PERMISSIONS)]
```

Everyone or Authenticated users are denied to view the item (user is Everyone and is Authenticated):

```
[(Deny, Everyone, 'view')]
# OR
[(Deny, Authenticated, 'view')]
# OR
[(Deny, Everyone, ALL_PERMISSIONS)]
# OR
[(Deny, Authenticated, ALL_PERMISSIONS)]
```

User 'john' is explicitly allowed to see an item BUT access is denied to 'group1' to which user belongs (order of ACEs doesn't matter):

```
[
    (Allow, 'john', 'view'),
    (Deny, 'group1', 'view'),
]
# OR
[
    (Deny, 'group1', 'view'),
    (Allow, 'john', 'view'),
]
```

1.3 Helpers

class nefertari_guards.base.**ACLEncoderMixin**

Mixin which implements ACL encoding/decoding.

Used in sqla and mongo ACLField.

classmethod stringify_acl(*value*)

Get valid Pyramid ACL and translate values to strings.

String cleaning and case conversion is also performed here. In case ACL is already converted it won't change. Input ACL is also flattened to include a singler permission per AC entry.

Structure of result AC entries is: {'action': '...', 'principal': '...', 'permission': '...'}

1.3.1 CLI

count_ace: Count the number of documents that contain a particular ACE. Prints the count of objects with matching ACE, listed by type, in CSV format.

class nefertari_guards.scripts.count_ace.**CountACECommand**

Usage example $ nefertari-guards.count_ace –config=local.ini –ace='{"action": "allow", "principal": "user1", "permission": "view"}' –models=User,Story

update_ace: Find documents that contain a particular ACE and replace that ACE with another ACE.

class `nefertari_guards.scripts.update_ace.`**`UpdateACECommand`**

Usage example $ nefertari-guards.update_ace –config=local.ini –from_ace='{"action": "allow", "principal": "user1", "permission": "view"}' –to_ace='{"action": "deny", "principal": "user1", "permission": "view"}' –models=User,Story

1.4 Changelog

- #12: Added CLI scripts to count/update ACEs
- #13: Fixed a bug with collection items not being properly denied due to ACL inheritance
- #14: Reworked ACL filtering to support ES 2.x
- : Fixed a 400 error returned by elasticsearch aggregation queries with 'auth = true'
- : Initial release

n

A

C

N

S

U