

Developer Guide / Talos

1. Talos – the expressive authorization engine

- 1.1. The problem
- 1.2. A fluent interface
- 1.3. The Bouncer
- 1.4. Categories
- 1.5. Conclusion

2. Download and configuration

3. General usage

- 3.1. Accessing Talos
- 3.2. Containers
- 3.3. The Bouncer

4. Data model

- 4.1. Simple and Group permissions
- 4.2. Subjects
- 4.3. Secure Objects
- 4.4. Object Categories

5. Getting creative

Talos
Javadoc

Back to
developer guide
overview

Luigi
search

1. Talos – the expressive authorization engine

At some point or another, every application needs to handle authorization processes.

Most authorization engines require your application code to jump through loops to check and change access rights. You end up polluting your code with things that don't have anything to do with authorization.

Explain yourself

"Who are YOU?" said the Caterpillar.

This was not an encouraging opening for a conversation. Alice replied, rather shyly, "I-I hardly know, sir, just at present — at least I know who I WAS when I got up this morning, but I think I must have been changed several times since then."

"What do you mean by that?" said the Caterpillar sternly. "Explain yourself!"

"I can't explain myself, I'm afraid, sir" said Alice, "because I'm not myself, you see."

"I don't see," said the Caterpillar.

Talos provides expressiveness to authorization code. It looks almost like english. Sure, it's not buzzword-compatible, but it's clean. You can check access rights and change them in single-line commands, directly from the code, without dealing with external XML, policy files, and classpath. See for yourself.

1.1. The problem

Talos is an authorization engine designed for easy and efficient management of access rights. All the work is done using an intuitive API that resembles a domain-specific language.

Authorization refers to the process of deciding if **someone** can **do something** on **something**:

- **Bob** wants to **feed Mingau the cat**
- **Win32.mm** wants to **send** and **receive** over the **network**.
- **Email from tminc@uol.com.br** wants to **be stored** on **the inbox**.

1.2. A fluent interface

Talos uses a Java API to store authorization data in a relational database. The sentences above translate to:

```
if (talos.withSubject("Bob")
    .andObject("Mingau")
    .isAllowed("feed")) {
    // do stuff
}
```

At any time, we can allow or deny stuff on many objects and many subjects:

```
talos.withSubjects("Bob", "Maggie", "John", "Denise", "Andy")
    .andObjects("Mingau", "Spotty", "Kitty")
    .grant("feed");

talos.withSubject("Win32.mm")
    .andObject("Network")
    .revoke("send", "receive", "listen");
```

Of course, Talos needs to know these elements first. Let's say Lisa comes to live with Bob and the others. She also needs to feed the cats. I can do it all at once (not that I would always want to):

```
talos.createSubject("Lisa")
    .andObjects("Mingau", "Spotty", "Kitty")
    .grant("feed");
```

1.3. The Bouncer

If I have to check many permissions, I can use the *bouncer*:

```
Bouncer bouncer = talos.withSubjects("Lisa", "Maggie")
    .andObject("Mingau");

if (bouncer.isAllowed("feed")) {
    // feed the cat
} else if (bouncer.isAllowed("stroke")) {
    // stroke the cat
}
```

Sometimes I need to ask Talos about the subjects (the **who** part) or the objects (the **what** part) that can do something with the permissions. Let's say I want to know who can feed *or* stroke Spotty and Kitty:

```
Collection<String> names = talos.withObjects("Spotty", "Kitty")
    .andAnyPermissions("feed", "stroke")
    .listNames();
```

Printing names would give me Bob, Maggie, John, Denise, Lisa, Andy.

I might also want to know which cats Maggie can stroke *and* feed:

```
Collection<String> names = talos.withSubjects("Maggie")
    .andAllPermissions("feed", "stroke")
    .listNames();
```

1.4. Categories

If a new cat comes along, I don't want to need to grant all permissions all over again. I will define a category and grant rights accordingly. Andy and John will be responsible for cats by default:

```
talos.createCategory("cats")
    .addObjects("Spotty", "Kitty", "Mingau");

talos.withSubjects("Andy", "John")
    .andCategory("cats")
    .grant("feed", "stroke");
```

This means that if I add a new cat, they can feed it:

```

talos.createObject("Caramel")
    .addCategories("cat");

Bouncer bouncer = talos.withSubjects("John")
    .andObject("Caramel");

Collection<String> names = bouncer.listNames();

```

I know that `names` contains at least `feed` and `stroke`, and that `bouncer.isAllowed("feed")` will return `true`. When querying, I might also restrict objects by category, so I am sure I'm asking about which *cats* Maggie can take care of:

```

Collection<String> names = talos.withSubjects("Maggie")
    .andCategory("cats")
    .andAllPermissions("stroke", "feed")
    .listNames();

```

1.5. Conclusion

Talos allows the programmer to express their authorization requirements concisely with minimal operational overhead. And *that* is, in our opinion, how access rights should be managed.

2. Download and configuration

You can download Talos2 from sourceforge. If you intend to use Ivy, you can declare a dependency on Talos2 using the following line in your `ivy.xml`:

```
<dependency org="os-kit" name="talos2" rev="0.5" conf="build,default" />
```

Configuring your environment

You need to configure the Hibernate access, as per the Hibernate configuration documentation. The mappings are configured in the `hibernate.cfg.xml` file. This is how it looks like for Talos 0.5:

```

<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>

        <mapping class="org.talos.model.Capability"/>
        <mapping class="org.talos.model.Category"/>
        <mapping class="org.talos.model.GroupPermission"/>
        <mapping class="org.talos.model.Permission"/>
        <mapping class="org.talos.model.SecureObject"/>
        <mapping class="org.talos.model.SimplePermission"/>
        <mapping class="org.talos.model.Subject"/>

    </session-factory>
</hibernate-configuration>

```

2. General usage

Talos has a very small set of operations, all dealing with access rights. You can check access rights or permissions, in order to decide if some business logic can be executed by someone. And you can grant or revoke permissions to someone.

To make things easier to write and to explain, there are usually two steps you must perform:

- open a Talos instance
- select the elements you're manipulating. This means choose one or more subjects and one or more secure objects (or, alternatively, some subjects and some categories).
- grant, revoke or check permissions on them.

3.1. Accessing Talos

Talos uses Hibernate to access the database. You can instantiate a `SessionFactory` any way you wish and provide it to the `TalosFactory`'s constructor. Or you can call `TalosFactory.createDefault()` to get a new instance. We recommend that you use an IoC container, but it isn't required in any way.

Each time you want to access the repository, you must open a new Talos instance from the factory, and close it at the end:

```
// open a new instance
Talos talos = factory.open();
// ... do something ...
// commit any changes you've done
talos.commit();
// close it when finished
talos.close();
```

Notice that you only need to call `commit()` if you're going to persist things. Otherwise, just call `close()` and Talos will take care of the cleanup.

What do you do with the Talos instance? Build containers, of course!

3.2. Containers

In order to grant, revoke or check permissions with Talos, you need to select a subset of the whole data to work with. Containers are objects that represent your selection. Some examples:

```
SubjectContainer sc = talos.withSubjects("mom", "dad", "Dave");
ObjectContainer oc = talos.allObjects();
CategoryContainer cc = talos.createCategory("music");
PermissionContainer pc = talos.createSimplePermissions("eat", "drink", "be merry");
```

3.2.1. Management operations

Containers usually perform management operations on the selected data. For example, after creating the category tools, you might want to add some data to it:

```
CategoryContainer cc = talos.createCategory("tools");
cc.addObjects("screw driver", "hammer", "plier");
```

Or let's say you want to delete some permissions:

```
pc.remove();
```

The containers also provide a few collection methods such as `contains(String)`, `list()` and `isEmpty()`. Refer to the javadoc of the Container interface for more details.

3.2.2. Combining containers

Once you select some subjects and some secure objects, you can grant, revoke or check access rights on them. The following are equivalent ways to do the same:

```
// first way:
talos.withSubjects("Chuck Norris")
    .andCategories("people")
    .grant("round kick");
// second way:
talos.withCategories("people")
    .andSubjects("Chuck Norris")
    .grant("round kick");
// third way:
SubjectContainer sc = talos.withSubjects("Chuck Norris");
CategoryContainer cc = talos.withCategories("people");
PermissionContainer pc = talos.withPermissions("round kick");
sc.andCategories(cc.list()).grant(pc.list());
```

Chuck Norris round kicking isn't funny, but I think the message gets through.

3.3. The Bouncer

Talos allows you to grant, revoke and check access rights once you have a bouncer. to get your hands on one, you need to combine containers:

```
Bouncer bouncer = talos.withSubjects("admin")
    .andObjects("one", "two", "three");
```

This bouncer can then be used to change permissions for the subject "admin" on the three objects. The

bouncer can:

- `isAllowed()` – checks one or more permissions
- `grant()` – grants/allows one or more permissions
- `revoke()` – revokes/denies one or more permissions
- `revokeAll()` – revokes all permissions currently granted to the subjects
- `changeTo()` – revokes current permissions and grant new ones
- `listNames()` – gets the names of the permissions currently directly granted to the subject
- `listExpanded()` – gets the names of the permissions directly granted to the subject, plus any permissions that might have been inherited from categories or group permissions.

`bouncer.isAllowed("delete")` is semantically equivalent to `listExpanded().contains("delete")`.

3.3.1. Permission inheritance

Consider the table below. Each entry defines a subject, either an object or a category, and a list of permissions.

Subject	Object	Category	Permissions
Andy	Mingau	–	Andy Kitty – stroke
Andy	–	cat	feed

Assuming that Kitty belongs to the category cat, the following sentence is true:

```
talos.withSubject("Andy")
    .andObject("Kitty")
    .isAllowed("feed", "stroke");
```

Andy is allowed to feed and stroke Kitty because each object inherits the permissions of the categories they belong to. The bouncer also knows if any additional permissions are implied by the group permissions in the permission set too, so you always get the full list.

Talos fetches the information it in the most efficient fashion, using a big (yet simple) Hibernate query. The results are cached, so if you need to know if Andy can feed or stroke a cat, you can use:

```
bouncer bouncer = talos.withSubject("Andy")
    .andObject("Kitty");

if (bouncer.isAllowed("feed") || bouncer.isAllowed("stroke")) {
    display("Kitty is happy!");
}
```

4. Data model

Talos handles with the following entities:

Secure Object

the target or instrument of an operation, usually data, a process or a service

Subject

The entity that executes operations

Permission

A token that allows an operation to be executed

Category

a collection of secure objects that share a common meaning.

We'll describe each one in the following sections.

4.1. Simple and Group permissions

Permissions are uniquely identified by their name, but exist as entities in the system. There are two kinds of permissions:

- Simple Permission: it just contains the name of the operation that it allows. Usually, they have verbs for names, such as "write", "delete", "drink", "call", "open".
- Group Permission: works just like a simple permission, but additionally implies other simple permissions.

One use for group permissions is to allow several simple permissions to be granted at once. Let's say an application has the typical "read", "write", "delete" simple permissions. I can instruct Talos to consider all of them by the name of "manage":

```
talos = factory.open();
talos.createGroupPermissions("manage");
talos.createSimplePermissions("read", "write", "delete")
```

```
.addToGroup("manage");
talos.commit();
```

Notice that if I grant "manage" to a subject, it will end up with four permissions, the group permission plus all its implied permissions. Group permissions might come in handy, but don't forget that they are regular permissions as well. As such, merely granting read, write and delete access will not automatically grant "manage".

4.1.1. Single-level nesting

Notice you can't add group permissions to group permissions, thus building a permission tree. Some engines provide that. We chose to only support one nesting level because that can be efficiently implemented over a relational database. Besides, we think this feature is rarely needed: if you just need to group permissions, store an array of names somewhere.

```
public static final String[] MANAGE = {"read", "write", "delete"};
// somewhere else:
talos.withSubject(aSubject)
    .andObject(anObject)
    .grant(MANAGE);
```

4.2. Subjects

Subjects, as other Talos entities, are identified by a name. This name must be unique in respect to other subjects. Apart from that, there isn't much to talk about them.

Talos can manipulate several subjects in one operation, and it assumes you are talking about all of them. The code below illustrates that:

```
boolean allowed = talos.withSubjects("Andy", "Lisa")
    .andObjects("Kitty")
    .isAllowed("feed");
```

Talos will check if both Andy and Lisa have the specified permissions on the target object. Similarly, `grant()` and `revoke()` also operate on all given subjects.

4.3. Secure Objects

Talos identifies secure objects by their name, as you've probably noticed. The name must be unique in respect to other objects, and it has no meaning at all, or relation to other entities. Just because there's a subject called "frog", one cannot assume anything about an object with the same name. It might exist or not, and if it exists, it might have a completely different meaning. The application is responsible for giving meaning to these names. Talos has only one job: managing access rights.

As with subjects, several objects can be manipulated in the same operation. Again, Talos assumes you are talking about all of them. The code below illustrates that:

```
boolean allowed = talos.withSubjects("Mary")
    .andObjects("Mingau", "Kitty")
    .isAllowed("feed");
```

The variable will be true if Andy has permission to feed both cats. If several permissions are checked, he must have all of them for all of the objects.

4.4. Object Categories

Most applications don't grant access rights to each individual object. This operation would be costly and can be avoided by the usage of categories. Categories are conceptual groupings of objects. Each category's name must be unique in respect to other categories.

In the overview, we add all cats to a category "cats", and then can grant access rights to the categories and have them inherited by the objects belonging to them.

```
talos.createCategory("cats")
    .addObjects("Mingau", "Kitty", "Spotty");
talos.withCategory("cats")
    .andSubject("Andy")
    .grant("feed");
```

Now Andy can feed all cats. If later I add a new cat and add it to the category, Andy will be able to feed it too. Of course, categories can be used more creatively.

5. Getting creative

How would we implement a friends list like the one in Flickr? Suppose I post some photos and I want to mark some users as friends and some pictures as "friends-only".

Since this refers mostly to authorization, we would be better off storing all this information in Talos. We'll create two categories:

- public photos keeps track of all photos in the system
- friends of <username> is created each time a new user is added, and keeps track of his photos

Each time I create a new user, we grant them rights to public photos. Each time a user marks (or unmarks) someone as a friend, we grant (or revoke) access rights on the category:

```
public void markAsFriend(String friendName, boolean marked) {
    String actor = authentication.getCurrentUser();
    Bouncer bouncer = talos.withSubject(friendName).andCategory("friends of " + actor);
    if (marked) {
        bouncer.grant("view", "comment");
    }
    else {
        bouncer.revokeAll();
    }
}
```

What happens when user "max2006" posts a new photograph?

- a secure object for the photo is created, let's say "pic1234";
- he is granted direct access to the picture;
- if the photo is public, it gets added to the category public photos;
- if it's not public but marked as friends-only, it gets added to his friends list

The logic looks like this:

```
public void grantRights(String userName, String photoName, boolean public, boolean friends) {
    Talos talos = talosFactory.open();
    // 1. create object
    ObjectContainer container = talos.createObject(photoName);
    // 2. grant direct access
    container.andSubject(userName).grant("view", "delete", "modify");

    // 3. add to category if public
    if (public) {
        container.addCategory("public photos");
    }
    // 4. add to friends list if marked friends-only
    else if (friends) {
        container.addCategory("friends of " + userName);
    }
    talos.commit();
}
```

This is better than storing authorization attributes directly in the model because it simplifies authorization checks:

```
public void viewPhoto(String photoName) {
    String actor = authentication.getCurrentUser();
    if (talos.withSubject(actor).andObject("photoName").isAllowed("view")) {
        // show picture
    }
}
```



Please send us comments, questions, criticism!