Custom authentication modules

Jochen Kressin · 2018-12-11

Search Guard is already capable of integrating with several authentication and authorisation backends like LDAP/AD, Kerberos, OpenId and JWT. Moreover, we also allow to integrate with any other system required by our customers. In this post we present a demo of custom authentication and authorisation module that fetches users and roles from a relational database.

Custom use case

Assume, we have a MySQL with the following users and user_roles tables:

```
CREATE TABLE users (username VARCHAR(50), hash VARCHAR(200));
CREATE TABLE user_roles (username VARCHAR(50), role VARCHAR(50));
```

We insert a username with a hashed password:

```
INSERT INTO users VALUES (
  'hr_employee',
  'd74ff0ee8da3b9806b18c877dbf29bbde50b5bd8e4dad7a3a725000feb82e8f1'
);
```

which is SHA256 hash of “pass”. We also insert roles:

```
INSERT INTO user_roles values ('hr_employee', 'sg_kibana_user');
INSERT INTO user_roles values ('hr_employee', 'sg_readall');
```

At the end, we would like to create custom authentication backend, that executes a query to fetch a user:

```
SELECT hash FROM users where username = ?
```

Then it will compare the hash from the database table with a hash of a password provided by a user. If they match, it will fetch user's roles with:

```
SELECT role FROM user_roles where username = ?
```

General rules for custom modules

Based on the Search Guards custom modules documentation, we create a maven project with two provided dependencies: search-guard-6 and elasticsearch.

```
<dependency>
  <groupId>com.floragunn</groupId>
  <artifactId>search-guard-6</artifactId>
  <version>${searchguard.version}</version>
  <scope>provided</scope>
</dependency>

<dependency>
  <groupId>org.elasticsearch</groupId>
  <artifactId>elasticsearch</artifactId>
  <version>${elasticsearch.version}</version>
  <scope>provided</scope>
</dependency>
```

We will need to create two classes that implement the following interfaces:

- AuthenticationBackend
- AuthorizationBackend

The common method for both interfaces is *getType*, which just returns a telling name for the module and which is helpful for logging and parsing the logs.

Both classes need to implement a constructor of a form:

```
public RdbmsAuthenticationBackend(final Settings settings, final Path configPath)
```

The instance of *Settings* allows us to grab configuration settings from sg_config.yml:

sg_config.yml:

```
authc:
  rdbms_authc_domain:
  ...
  authentication_backend:
  type: com.floragunn.custom.rdbms.RdbmsAuthenticationBackend
  config:
    jdbc_url: 127.0.0.1
```

RdbmsAuthenticationBackend:

```
jdbcURL = settings.get("jdbc_url");
```

The common use case for custom auth/auth modues is to connect external services. Thus, we need to take care of the Elasticsearch Java Security permissions. This requires to put code within *AccessController.doPrivileged* method, as mentioned with Elasticsearch documentation.

Implementing interfaces

Only a single method needs to be implemented within each interface. Authentication requires an authenticate method with a signature:

```
public User authenticate(AuthCredentials credentials) throws ElasticsearchSecurityException
```

Username and password are extracted from credentials. In our demo use case, we query the database to see if a user exists. Then we compare secured hash from database with the one provided by user. If they do match, we return User object:

```
return new User(credentials.getUsername());
```

If they do not match, or the user does not exist in database, we throw an ElasticsearchSecurityException.

Authorisation requires implementation of:

```
public void fillRoles(User user, AuthCredentials authCredentials) throws
ElasticsearchSecurityException
```

We fetch the roles from a database and add it with

```
user.addRoles(List<String>)
```

Full implementation of demo authenticator and authorisation can be found on Github.

Deployment

This only requires:

- building a JAR file
- copying it into plugins/search-guard-6/

- restarting the cluster

It is possible that your custom implementation depends on a JAR that is already on the cluster. This may result with:

```
Caused by: java.lang.IllegalStateException: failed to load plugin search-guard-6 due to jar hell
```

In our demo scenario, it may be that Guava that is already on Elasticsearch classpath. The solution is to change the scope of the dependency in *pom.xml* to *provided*.

Live demo

We add entries to sg_config.yml:

```
authc:
  custom_authc_domain:
  enabled: true
  order: 0
  http_authenticator:
    type: basic
    challenge: false
  authentication_backend:
    type: com.floragunn.custom.rdbms.RdbmsAuthenticationBackend
    config:
      jdbc_url: 127.0.0.1
      jdbc_user: dbuser
      jdbc_password: pass
      jdbc_database: searchguard
```

and:

```
authz:
  custom_authz_domain:
  enabled: true
  authorization_backend:
    type: com.floragunn.custom.rdbms.RdbmsAuthorizationBackend
    config:
      jdbc_url: 127.0.0.1
      jdbc_user: dbuser
      jdbc_password: pass
      jdbc_database: searchguard
```

Do not forget to reload cluster config with the *sgadmin* command. Then we check the *authinfo* endpoint with:

```
curl -k 'https://localhost:9200/_searchguard/authinfo?pretty' -u hr_employee:pass
```

{

```
  "user" : "User [name=hr_employee, roles=[sg_kibana_user, sg_readall], requestedTenant=null]",
  "user_name" : "hr_employee",
  "user_requested_tenant" : null,
  "remote_address" : "127.0.0.1:53782",
  "backend_roles" : [
    "sg_kibana_user",
    "sg_readall"
  ]
  ...
}
```

This means we successfully authenticated a user based on credentials stored in relational database. The user has been added to *sg*kibana*user* and *sg*readall_ roles as expected which ends our demo.

Where to go next

- Head over to our official documentation for custom modules
- Check out the sample code for this blogpost on GitHub
- Any questions? Ask them in our forum.

Image: shutterstock / kmlmtz66

This article was published on the Search Guard blog: https://search-guard.com/blog/elasticsearch-custom-auth-modules/
Search Guard is Security and Alerting for Elasticsearch and Kibana — search-guard.com · Start a free trial