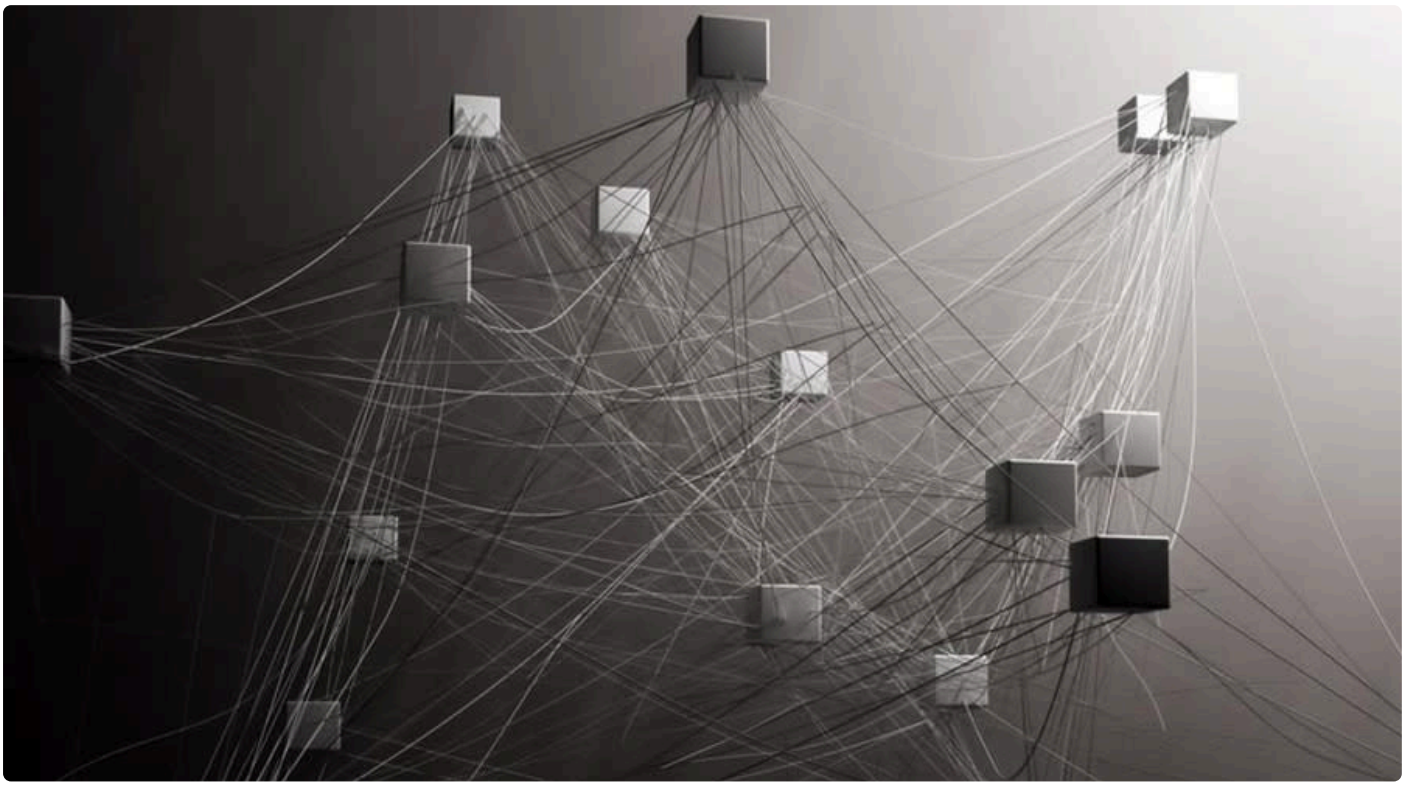
Transport client authentication and authorization

Jochen Kressin · 2016-11-13



In the last article in this series, we have set up a transport client, configured it with a client certificate, and defined the allowed roles and permissions for this certificate. This was based on the Distinguished Name of the certificate. However, this approach is limited to having just one single user. In this article, we'll show how to pass additional user credentials to Search Guard, in order to have full-fledged access control on the transport level.

Prerequisites

If you have not already done so, please follow the steps in the last article to set up basic transport layer access to a Search Guard secured cluster.

If you have followed the steps accordingly you should be able to connect to the cluster on the transport level using the *spock* client certificate.

We used the Distinguished Name of this certificate to assign it to the role *sg*readall_:

```
sg_readall:
  users:
    - user1
    - user2
    - 'CN=spock,OU=client,O=client,L=Test,C=DE'
```

This allowed us to set up permissions for this role, and apply them to any request with the *spock* certificate. This is all good, but imagine you want to implement an application that accesses a Search Guard secured cluster, and access should be granted based on the credentials of the logged in user, and not the certificate. This is possible with Search Guard, and you can have the same fine-grained access control in the transport level as you have on the REST level. Here's how to do it:

Basic principle

The basic principle is very simple. You will need a client certificate that is signed by the root CA and thus is trusted by Search Guard. Use only client certificates, and not server or admin certificates (see below). This certificate has to be sent with each request to allow connections on the transport level.

Next, define a user identified by the Distinguished Name of the certificate. The user does not need to have any roles or permissions, it just needs to be known to Search Guard.

Last, send the credentials of the user that you want to use with each transport request. We will send it as a header that resembles an HTTP Basic Authentication header closely.

Search Guard will then use this credentials to apply roles and permissions, just like for any other request.

Defining a transport user based on the client certificate

Since we do not need any roles or permissions based on the Distinguished Name of the certificate anymore, we'll remove it from the role *sg*readall_:

```
sg_readall:
  users:
    - user1
    - user2
```

However, we still need a user identified by the DN. If you use the internal user database, add:

```
CN=spock,OU=client,O=client,L=Test,C=DE:
  hash: "_transport_only"
```

Of course any other authentication backend, like LDAP, will also work. This makes sure that Search Guard accepts connections identified by this certificate, and is similar to the setup we used in the last article.

Adding username and password to the request

Credentials on the transport level work very similarly than credentials provided via HTTP Basic Authentication. In fact, we use the very same format and mechanism, meaning that we

- Add an "Authenticate" header to each request
- Which contains the Base64 encoded username and password separated by a colon

A typical HTTP Basic header on the REST level looks something like this:

```
Authorization: Basic QWxhZGRpbjpPcGVuU2VzYW1l
```

The part after *Basic* is the Base64 representation of

```
Aladdin:OpenSesame
```

Since HTTP is stateless, we need to provide this Authentication header for each request. Typically, your browser will take care of this.

Basic Authentication on transport level

Now, we simply apply the same principle to the transport client. Concatenate the username and password with a colon, Base64 encode it and add it as header to the request:

```
TransportClient client = ...

For ES2.x:
client.prepareGet(<span class="s2">"index", <span class="s2">"type", <span
class="s2">"id").putHeader(<span class="s2">"Authorization", <span class="s2">"Basic
"+encodeBase64(<span class="s2">"username:<span class="s2">password")).get();

For ES5.x:
client.threadPool().getThreadContext().putHeader("Authorization", "Basic
"+encodeBase64("username:password"));
tc.prepareGet(...).get();
```

Search Guard will extract the user credentials from the request, validate them and apply roles and permission settings. From this point on, there is no difference between authentication on transport and REST level. This means you can implement access control based on indices and types, and also apply document- and field-level security.

Since also the transport protocol is stateless, you need to make sure to send the *Authentication* header for each request.

That's all we need to do, and we can now use all features of Search Guard also on the transport level!

Different types of certificates

There are different types of certificates that Search Guard uses:

- Node/Server certificate: Used by the cluster nodes on transport level. Node certificates are identified by a special SAN entry in the certificate. Please see chapter Server certificate OID in the documentation.
- Admin certificate: Used on transport and/or REST level. Identifies a Search Guard administrator. Admin certificates are identified by their Distinguished Name. This DN needs to be configured in elasicsearch.yml. Please see chapter Configure the admin certificate in the documentation.
- Client certificate: Any other certificate that is signed by the root CA

If you are using transport level authentication, please use client certificates only. Server certificates and admin certificates have full access to the cluster, and Search Guard will not apply any security checks if they are provided!

Summary

In order to use transport level authentication:

- Set up a transport client, and configure it to use a client certificate when talking to Elasticsearch
- Configure the Distinguished Name of the certificate as a Search Guard user. This user does not need any login credentials or roles since it is already identified by its certificate
- Send a *Basic Authentication* header with each request. This header contains the username and password and want to use for this request
- Search Guard will apply all security checks for the provided credentials, as if they would have been provided on the REST level
- This means you can implement access control based on indices and types, and also apply document- and field-level security on the transport level.

Image: Mopic / Shutterstock

This article was published on the Search Guard blog: https://search-guard.com/blog/transport-client-authentication-authorization/