Elasticsearch client certificate authentication

Cliff Stayley · 2019-02-06

Search Guard supports TLS (Transport Layer Security) by design. We require secured node to node transmission and provide Elasticsearch API over HTTPS. TLS relies on certificates what we have explained in the Introduction to TLS certificates article. Moreover, we have shown a demo of generating certificates with openssl in another post.

The most common use case for TLS is the ability for a client to verify the identity of a web server it connects to. For example, a web browser opening the search-guard.com website verifies if it is really connecting to the Search Guard webpage. The concept may be also used the opposite way - a client certificate can be used to authenticate the client on the server side. This can be especially useful for organisations that have implemented and rely on their own PKI infrastructure.

Client certificate authentication overview

- A client obtains a certificate that is signed with a root certificate authority trusted by the server
- The client owns its private key used to generate the certificate and the certificate that contains the public key
- When the client connects to the server it sends its certificate and some data encrypted with the private key
- The server decrypts data with the public key. This makes sure that client, who has sent a request, is the owner of the certificate as generating the request required the private key

- The certificate describes its owner (distinguished name) and the server can trust it as it has been signed with a root CA they both trust

Generating client certificate

We start with creating a certificate configuration file describing the client:

```
[ req ]
prompt              = no
distinguished_name  = client
```

[client]

```
            commonName = hr_employee
           countryName = DE
      organizationName = Demo-SearchGuard
organizationalUnitName = demo-cluster
   stateOrProvinceName = Berlin
```

Then we create client private key and a certificate signing request with:

```
openssl req \
  -newkey rsa:4096 \
  -keyout client_key.pem \
  -out client_cert.csr \
  -days 365 \
  -nodes \
  -config client_cert
```

In the next step, we sign *hr*employee_*'s* request with the private key of the root certificate authority. If you follow the setup shown in this post, you can do it with this command:

```
openssl x509 \
  -req \
  -in client_cert.csr \
  -CA root-ca.pem \
  -CAkey key.pem \
  -CAcreateserial \
  -out client_cert.pem \
  -extfile client_cert \
  -days 365
```

The arguments provided include root certificate (*root-ca.pem*) and its private key (*key.pem*). As an output, we get a signed client certificate in a file *client*cert.pem_.

When we finish the setup, we will send a request:

- containing a client certificate, stored in *client*cert.pem_
- and signed with the client private key stored in *client*key.pem_

Search Guard configuration

Search Guard configuration requires the following changing *elasticsearch.yml* with:

```
searchguard.ssl.http.clientauth_mode: OPTIONAL
```

That allows the Search Guard plugin to fetch the client certificate from the HTTP request. Instead of OPTIONAL, the other possible values are REQUIRE or NONE.

- *NONE*: Search Guard will not pick up any TLS client certificate from the HTTP request
- *OPTIONAL*: Search Guard will pick up a TLS client certificate if present
- *REQUIRE*: Search Guard requires a client TLS certificate. If none is provided, the request will not be accepted

Afterwards, the cluster needs to be restarted.

Next, we will configure an authentication domain that supports client certificate authentication:

```
clientcert_auth_domain:
    http_enabled: true
    transport_enabled: false
    order: 1
    http_authenticator:
      type: clientcert
      config:
        username_attribute: cn #optional, if omitted DN becomes username
      challenge: false
    authentication_backend:
      type: noop
```

The important setting is the *username*attribute: *This defined which part of the distinguished name of the client certificate should be used as the username. For example, we configured the* cn_ part of the certificate to denote the username. The distinguished name of our client certificate is:

```
CN=hr_employee,OU=demo-cluster,O=Demo-SearchGuard,C=DE
```

With the *username*attribute *configuration in* sg*config.yml*, Search Guard extracts the username from the *cn* field, so it will be *hr*employee_ in our example.

Role mapping

We also need to add or modify *sg*roles*mapping.yml*, so that the user will be mapped to a Search Guard role. In our demo, we will map the user to the *sg*readall_ role:

```
sg_readall:
  users:
    - hr_employee
  backendroles:
    - ...
```

Do not forget to reload cluster configuration with the *sgadmin* CLI tool to apply the changes.

Demo

The last thing to do to test the configuration is to request the *authinfo* endpoint with curl. We will use curl it with *--cert* and *--key* options so curl will send the TLS client certificate with the request:

```
curl -k 'https://localhost:9200/_searchguard/authinfo?pretty'   \
    --cert client_cert.pem \
    --key client_key.pem
```

This returns a JSON string with *hr*employee *user's data and the* sg*roles* we have configured:

```
{
  "user" : "User [name=hr_employee, roles=[], requestedTenant=null]",
  "user_name" : "hr_employee",
  "user_requested_tenant" : null,
  "remote_address" : "[::1]:59110",
  "backend_roles" : [ ],
  "custom_attribute_names" : [ ],
  "sg_roles" : [
    "sg_readall"
  ],
...
```

Where to go next

- Refer to the Search Guard documentation on TLS client certificate authentication
- Read our other blog posts on TLS and TLS certificates
- Use our TLS tool to generate TLS certificates
- If you have questions, ask them on the Search Guard forum

Image: shutterstock / pockygallery

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