Proxy authentication using TLS certificates

Aki Abramowski · 2020-02-24

Search Guard supports proxy authentication since the very first release. If you are running Elasticsearch or Kibana behind a Proxy, for example nginx, you can delegate authentication and authorization to the proxy. After the proxy has successfully authenticated the user, it adds the username and (optional) the user's roles as HTTP headers to each request to Elasticsearch. Search Guard picks then up the authenticated user's credentials, and assigns Search Guard roles and permissions.

Security considerations

This approach has two flaws: The request from the proxy to Elasticsearch can be sniffed and tampered with. And anyone can modify the headers of an HTTP request and add any username and roles to it.

The first problem is easy to fix: Configure TLS encryption on the REST layer. Now the requests cannot be sniffed or modified anymore. Done.

The second one is a bit more complicated. How does Search Guard know that the request that carries the user name actually comes from a trusted proxy, and not from some malicious attacker?

IP based trust

Until now, there was only one solution: You had to tell Search Guard the list of IPs that should be deemed trustworthy. So basically a list of your proxy IP(s), for example:

```
sg_config:
  dynamic:
    http:
      xff:
        enabled: true
        internalProxies: '172.16.0.203' # nginx proxy IP
```

Here, only requests from the IP 172.16.0.203 are treated as trustworthy requests, and only those are accepted for proxy authentication.

Although you can specify multiple IPs and also use wildcards and regular expressions, the flexibility of this approach is limited..

A better approach

> IP based approaches are not playing well in dynamic environments.

IP based verification works well if the proxy IPs are static or do not change very frequently. But how would you use this approach on a completely dynamic environment like Kubernetes? Pods are created and destroyed dynamically, and the IPs are in most cases not known before a pod is started. This is where IP based verification falls short.

The new proxy module which will replace the current module starting with Search Guard 8 now also supports TLS client certificate verification.

TLS client authentication

How does it work? Basically like any other TLS connection: The client, which is the nginx proxy in our example, sends a TLS client certificate for every request to the upstream server, in our case Elasticsearch / Search Guard.

Search Guard validates the client certificate it received from nginx against one or more configured root CAs. Only if the certificate is valid, and only if the DN of the certificate matches a list of trusted DNs, the proxy is deemed trustworthy.

This approach provides much more flexibility in dynamic setups. You do not need to know any IPs of your proxy servers upfront. Instead, you create one or more TLS client certificates to verify trusted proxies. The client certificates can be rolled out during infrastructure provisioning.

Example configuration

Prerequisites

Let's assume you have already installed Search Guard on Elasticsearch and hence have already generated TLS certificates. For example, by using the Search Guard TLS Tool.

You have configured the root CA (used to generate all other certificates) in elasticsearch yml, like:

```
searchguard.ssl.http.pemtrustedcas_filepath: root-ca.pem
...
```

And let's say the DN of your root CA is:

```
CN=root.ca.example.com,OU=CA,O=Example Com.,DC=example,DC=com
```

As a next step, you use the same root CA and create one or more TLS client certificates that we will use on our proxy. There are no special requirements regarding those certificates.

In this example we will create just one certificate with the following DN:

```
CN=proxy.example.com,OU=CA,O=Example Com.,DC=example,DC=com
```

Configuration

Let's now use our newly created TLS certificate to authenticate our proxy:

First, we switch on TLS client certificate handling on the HTTP layer. We set the `clientauth_mode` to `OPTIONAL`, which means Search Guard will pick up and use TLS client certificates if present, but does not strictly require them. In elasticsearch.yml, add:

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

We can now enable the proxy authentication module in `sg_config.yml` and tell it to use TLS client certificates for validating our trusted proxies.:

```
sg_config:
    dynamic:
        authc:
            proxy2_auth_domain:
                http_enabled: true
                order: 0
                http_authenticator:
                    type: "proxy2"
                    config:
                        auth_mode: "cert"
                        user_header: "x-proxy-user"
                        roles_header: "x-proxy-roles"
                        roles_separator: ","
                        allowed_dn_s:
                            - "CN=proxy.example.com,OU=CA,O=Example Com.,DC=example,DC=com"
                authentication_backend:
                    type: "noop"
```

Let's step quickly over the configuration:

auth_mode: "cert" This tells the proxy module to use client certificates to identify trusted proxies, and not their IPs.

allowed*dn*s: This tells the proxy module that only TLS certificates with these Distinguished Names (DNs) are trusted. In this section we configure the DN that we used when generating the client certificate:

```
CN=proxy.example.com,OU=CA,O=Example Com.,DC=example,DC=com
```

You can specify an array of DNs, and you can also use wildcards and regular expressions here, for example:

```
CN=proxy-*.example.com,OU=CA,O=Example Com.,DC=example,DC=com
```

With this setting, any of the following DNs is deemed valid:

```
CN=proxy-1.example.com,OU=CA,O=Example Com.,DC=example,DC=com
CN=proxy-2.example.com,OU=CA,O=Example Com.,DC=example,DC=com
CN=proxy-3.example.com,OU=CA,O=Example Com.,DC=example,DC=com
...
```

user_header Defines the name of the HTTP header field where the username is stored.

roles_header Defines the name of the HTTP header field where the roles of the user are stored (optional)

roles_separator Defines how the roles in the roles_header HTTP header are separated. A roles String might look like:

```
it,devops,user
```

In this case you want to set the roles_separator to ",".

That's it!

nginx configuration

A very minimal nginx configuration might look like:

```
http {
  server {
    listen       8080;
    server_name  nginx.example.com;

    location / {
      proxy_pass https://elasticsearch.example.com:9200/;
      ...
      proxy_ssl_certificate     /etc/nginx/client.pem;
      proxy_ssl_certificate_key /etc/nginx/client.key;
      proxy_ssl_trusted_certificate /etc/nginx/root_ca.pem;
      ...
      proxy_set_header x-proxy-user admin;
      proxy_set_header x-proxy-roles admin;
    }
  }
}
```

This configuration tells nginx to send the client TLS certificate in `client.pem` to the upstream server for each request. For demo purposes, we have set the `x-proxy-user` and `x-proxy-roles` to a static value "admin". In real life this would be populated dynamically with the name and role(s) of the authenticated user.

The TLS certificate that nginx sends is first validated against the root CAs configured in your elasticsearch.yml. If this succeeds, the DN of the certificate is checked against the list of valid DNs in the Search Guard sg_config.yml configuration. If this also succeeds, the proxy is deemed trusted and the user credentials are extracted from the HTTP request.

Advanced settings

The new proxy module is of course backwards compatible with the existing module, so you can still use IP-based validation, or even a combination of IPs and certificates:

By setting **auth_mode** to **"both"** a trusted IP **and** a TLS certificate with a valid DN must be provided. If you don't need that extra security but have a setup where some requests can provide a certificate and others don't you can use the **"either"** mode. In that case the request is deemed valid if it has either a valid IP address **or** a valid certificate. For more information about such special cases take a look at our documentation.

Where to go next?

- Read about all configuration options in the Search Guard documentation
- Ask any questions on the Search Guard community forum
- If you have a feature request, head over to the feature request section on our forum

Image: shutterstock / Alexander Yakimov

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