# Don’t rely on proxies for Elasticsearch security

Jochen Kressin · 2017-05-27


When it comes to easy and affordable security for Elasticsearch, you might be tempted to rely on a “roll-your-own” security infrastructure based on proxies. This article shows why this is a bad idea, and why you should not use this approach for securing your sensitive data in production.

> tl;dr: Many organizations are using reverse proxies for securing their Elasticsearch infrastructure. Our view is that a proxy does not provide defense against a significant number of broad attack vectors, especially those originating internally. Although a proxy can add to the defense it is not enough on its own to meet any kind of security/risk profile for the Enterprise.

## Setup

In a typical setup, a reverse proxy (for example nginx) is placed in front of your Elasticsearch cluster. The proxy forwards all requests to Elasticsearch. It usually implements the following functionalities:

- It terminates TLS
  - Elasticsearch does not support TLS/SSL out of the box itself
- It authenticates the user
  - For example via simple Basic Authentication or advanced technologies like LDAP
- It implements access control based on the requested URL
  - The URL contains the index name and document type name in many, but not all cases

With this setup, it seems that you have secured your cluster sufficiently. The traffic is encrypted, the user has to authenticate, and you can allow or deny requests based on the request type, the index name and optionally the document type.

However, at a closer look, a setup like this is nothing more than "Security Theatre": You have the impression you are safe, while in reality, your cluster is open to attacks, and your data is still at risk. Here's why:

## Unencrypted HTTP traffic

Since TLS is terminated by the proxy, the complete traffic behind it is still HTTP. Which means it can be sniffed and tampered with. Some might argue that the Elasticsearch cluster is on a private network behind tons of firewalls and other security systems. So, the traffic cannot be accessed by a malicious hacker. Or can it?

> 60% of all attacks come from the inside

A case study by IBM revealed that 60% percent of all attacks come from the inside, not the outside. Which leaves your sensitive data at a high risk of being hijacked, even when it's surrounded by firewalls and VPNs.

## Unencrypted inter-node traffic

The statement above also applies to the traffic between the nodes in your cluster: It's unencrypted and insecure, which makes it very easy to access and modify it. Search Guard provides TLS encryption on both the HTTP and the transport layer at absolutely no cost, so there shouldn't be any reason not to use TLS to secure your traffic. TLS makes sure that no one can sniff or tamper your data.

## Transport client and cluster topology vulnerability

While a proxy can help to control what a client can do on the HTTP level, there is no way to restrict what a transport client can do. This also includes your cluster topology: Anyone can fire up a new Elasticsearch node, let it join the cluster and do malicious things.

## Composite request vulnerabilities

In most cases, the URL of the Elasticsearch call contains the index name, and optionally the document type, which can be used to implement security control. However, for some calls, the security relevant information is contained in the body of the request, not the URL. This makes it impossible to implement a sound security layer by analyzing the URLs alone. For example:

### Bulk API

The bulk API "makes it possible to perform many index/delete operations in a single API call". This means that a single API request can carry an arbitrary number of subrequests, with different actions on different indices.

> You have to inspect the request body as well

Without inspecting and analyzing this JSON payload, and matching it against the user's actual permissions, it is impossible to implement proper security checks. Search Guard resolves all affected indices and actions properly, also for bulk actions, and then applies security checks to each subrequest individually.

- (https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-bulk.html)

### Multi Get and Multi Search API

These APIs make it possible to submit multiple get or search requests on different indices in one call. As with the bulk API, the subrequests are contained in the request body. Search Guard checks each sub-request separately and allows/denies access accordingly. This is not possible with a proxy alone.

- (https://www.elastic.co/guide/en/elasticsearch/reference/current/multi-index.html)
- (https://www.elastic.co/guide/en/elasticsearch/reference/current/search-multi-search.html)

Kibana also makes heavy use of these composite requests, making it extremely difficult to set up solid security rules but still allow Kibana to work correctly. Search Guard supports Kibana out of the box.

## Delete-by-query and Update-by-query vulnerabilities

Again, the affected document types are implicitly contained in the JSON payload of the request, making it impossible to control by a proxy based solution alone.

- (https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-delete-by-query.html)

## Multiple indices and wildcard expansion vulnerabilities

Most Elasticsearch APIs that accept an index name as a parameter also support execution across multiple indices. You can specify multiple indices separated by a comma, in any order, like:

```
https://localhost:9200/index1,index2/_search
```

Elasticsearch also allows using wildcards, like:

```
https://localhost:9200/i*ex*/_search
```

Multiple indices and wildcards and also be combined:

```
https://localhost:9200/i*ex*,otherindex/_search
```

This makes it difficult or impossible to handle all cases correctly by a proxy. Search Guard resolves indices with wildcards to the actual index names, thus making sure that only permitted indices can be accessed.

- (https://www.elastic.co/guide/en/elasticsearch/reference/current/multi-index.html)

## Alias handling vulnerabilities

The index alias API "allow to alias an index with a name, with all APIs automatically converting the alias name to the actual index name". An index alias can also be mapped to multiple indices at once! So, before the request hits Elasticsearch and the alias name is expanded, how can you know which actual indices are affected by the request?

As with wildcards, Search Guard resolves any index alias to its actual index name(s), making sure permissions checks are applied correctly. This is done transparently to the user.

- (https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-aliases.html)

## Date math index name resolution vulnerabilities

Index names can also contain date math expressions, for easier handling of date-based index names. A request using date math expression could look like:

```
GET /<logstash-{now/d}>/_search
```

This is impossible to control with a proxy, while Search Guard uses the already expanded, actual index name to apply permission checks.

- (https://www.elastic.co/guide/en/elasticsearch/reference/current/date-math-index-names.html)

## Granularity of permissions – Document and Field level

A proxy based solution is limited to indices and document types. There is no way to implement access control on a finer level. Search Guards supports control down to the document and field level, so users only see documents and fields they are allowed to see.

## Missing audit logging capabilities hinders compliance

Audit logging cannot be implemented by a proxy alone. The proxy only sees the HTTP traffic, omitting the transport layer completely. It also does not see the actual index names when using aliases and wildcards. It cannot evaluate sub-requests contained in the HTTP body. For audit logging, proxies are pretty useless, making it hard to meet HIPAA, PCI DSS, FISMA or ISO compliance rules.

Search Guard adds audit logging capabilities directly to Elasticsearch, making it possible to log any activity on your cluster, including the actual, expanded index names, the submitted queries and subqueries, and any traffic on the transport layer.

## Additional attack surface

Having a proxy clearly enlarges the attack surface. A better approach is to harden your existing infrastructure with Search Guard rather than introduce a new attack surface. Attack surface is also directly related to porosity and proxies also increase porosity.

## Summary

Relying on a proxy-based security solution for protecting your sensitive data is not enough, and leaves your cluster open for all kinds of attack scenarios. By using Search Guard you can protect your data directly where it lives – in your Elasticsearch cluster.

Image: shutterstock / Gustavo Frazao

This article was published on the Search Guard blog: https://search-guard.com/blog/elasticsearch-proxy-security/