

Field anonymization

Cliff Staley · 2018-12-21



Elasticsearch is often used to store sensitive data. Moreover, a single index may store both sensitive and insensitive data that should be accessible for a broader audience. Search Guard allows you to implement fine-grained access control to these fields by using field level security and field anonymisation.

Search Guard provides two ways of controlling access to fields in a document:

- Whitelisting and blacklisting fields ([field-level security/FLS](#))
- Anonymising fields

Both of them are performed on the fly when executing a query. This means they do not require any data transformations or duplications at ingest time. The rules controlling field access can be changed at runtime without the need to change the underlying data.

In this post we show a demo of a field anonymisation and its capabilities. We will work on a sample log dataset to extract logs and count unique IP addresses.

Field anonymisation: Hashing and regular expressions

Search Guard provides two ways of anonymising field values: hashing and regular expressions.

Hashing

- Uses a (salted) hash function to replace the complete content of a field with a consistent hash
- E.g. replace the "firstname" and "lastname" fields of a document with a hash

Regular expressions

- Uses a regular expression to replace parts of a field value matching the expression with a constant value
- E.g. detect IP addresses in a field and replace them with "X.X.X.X"

In this first article of a two-part series we will use hashing.

Setup

We start with uploading a [sample log dataset](#) with the following curl call:

```
curl -k -XPOST 'https://localhost:9200/_bulk?pretty' \  
  --data-binary @logs.jsonl \  
  -H 'Content-Type: application/x-ndjson' \  
  -u admin:admin
```

and create a *logstash* index pattern in Kibana that aggregates the daily indexes.

Create index pattern

Kibana uses index patterns to retrieve data from Elasticsearch indices for things like visualizations.

☐ ☒ Include system indices

Step 1 of 2: Define index pattern

Index pattern

You can use a * as a wildcard in your index pattern.
You can't use spaces or the characters \, /, ?, ", <, >, |.

[> Next step](#)

✓ **Success!** Your index pattern matches **3 indices**.

logstash-2015.05.18

logstash-2015.05.19

logstash-2015.05.20

Rows per page: 10 ▾

Let us now create *logreader* user in *sg_internal_users.yml*

```
logreader:  
  hash: $2y$12$JKL4wtyT7Nx0szBqjjaEXOVGXWhLhRPNCAZppIvbvPmDmcXMbEo82  
  #password is: pass  
  roles:  
    - kibanauser  
    - sg_logs_anonymised
```

and a role in *sg_roles.yml* that masks all *ip** fields within an index:

```
sg_logs_anonymised:
  indices:
    'logstash*':
      '*':
        - READ
  _masked_fields_:
    - 'ip*'
```

Note that we need to use wildcard as the index contains two fields *ip* and *ip.keyword* to allow searching by exact values.

We then map Search Guard roles to backend roles by adding an entry in *sg_roles_mapping.yml*:

```
sg_logs_anonymised:
  backendroles:
    - sg_logs_anonymised
```

We also add salt for hashing in *elasticsearch.yml*:

```
searchguard.compliance.salt: <some-long-random-string>
```

and reload the cluster configuration:

```
./sgadmin.sh -cd ../sgconfig/ -icl -nhnv \
  -cacert ../../../../config/root-ca.pem \
  -cert ../../../../config/kirk.pem \
  -key ../../../../config/kirk-key.pem
```

That finishes the setup.

Testing

We can log into Kibana with the *logreader* credentials to see how the data looks like. The sample dataset contains logs from 2013, so do not forget to change the time range in Kibana:

New Save Open Share Reporting  Auto-refresh Last 15 minutes

Time Range

[Quick](#) [Relative](#) [Absolute](#) [Recent](#)

Today	Last 15 minutes	Last 30 days
This week	Last 30 minutes	Last 60 days
This month	Last 1 hour	Last 90 days
This year	Last 4 hours	Last 6 months
Today so far	Last 12 hours	Last 1 year
Week to date	Last 24 hours	Last 2 years
Month to date	Last 7 days	<u>Last 5 years</u>
Year to date		

Anonymised records should now look like this:

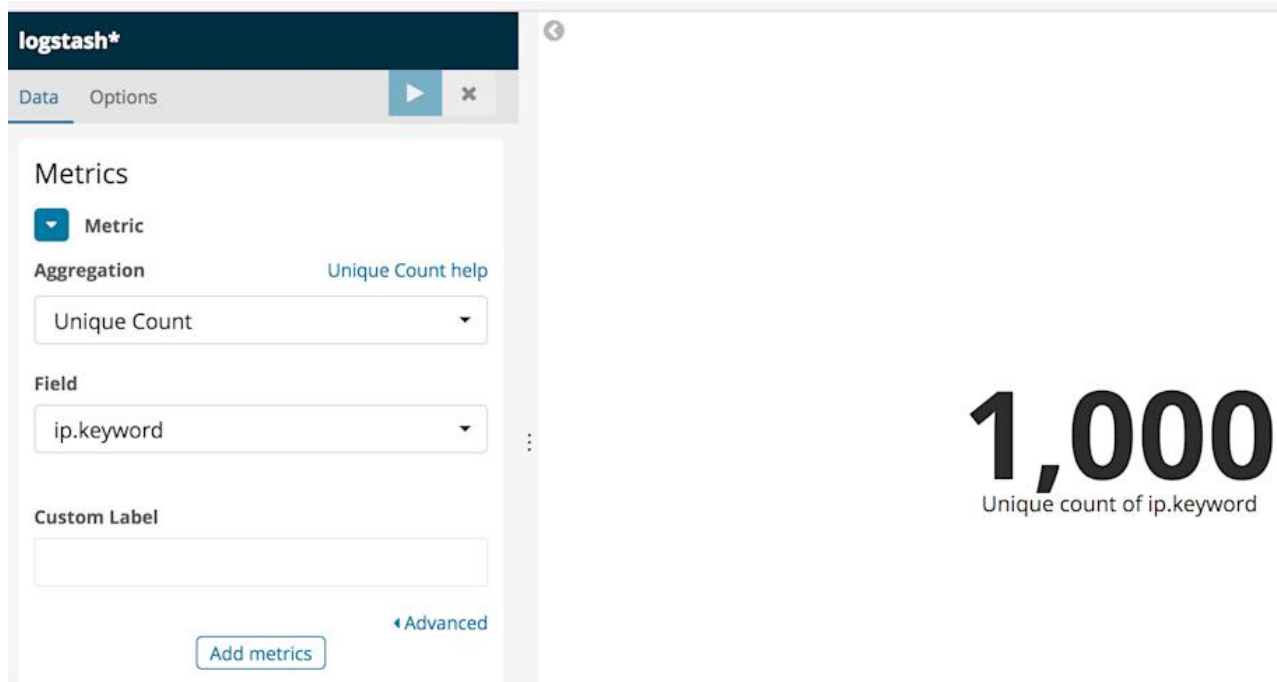
t	headings	   * aleksei-leonov , http://facebook.com/success/mercury-redstone-3
t	host	   * media-for-the-masses.theacademyofperformingartsandscience.org
t	ip	   * 16153db5f7d9c35c57325c20b1d72d8b7608b7608071e4276bbd4146d4bf1016

The *ip* field has been converted into hash on the fly for the *sg_logs_anonymised* role. We can also perform a quick cross-check and log in to Kibana with a role that does not have field anonymisation configured. For example, if you are using the Search Guard demo setup you can use the *admin* user. When viewing the log data with this user, the *ip* field is displayed in clear text.

Sometimes when changing configuration it may take some time before Kibana clears its caches. In this case, we encourage you to clear browser data or test with CURL:

```
curl -k -XGET 'https://#es-url#:9200/logstash-*/_search' \
-u logreader:pass | grep --color=auto "ip"
```

Let us now check how many unique IP addresses are within logs:



This query could not have been answered with field-level security alone and shows the advantage of field anonymisation in that specific case. Fortunately, Search Guard implements them both, so you can pick the one that suits best your needs or even mix them together.

Where to go next:

- Read the [official documentation on field anonymisation](#)
- Read our [post about document- and field-level security](#)
- Advanced field anonymisation using regular expressions (upcoming)
- If you have any questions, [ask them on our forum](#)

Image: shutterstock / [mr. graengkrai trusadee](#)

This article was published on the Search Guard blog: <https://search-guard.com/blog/elasticsearch-field-anonymization/>
Search Guard is Security and Alerting for Elasticsearch and Kibana — search-guard.com · [Start a free trial](#)