

You have **1** free member-only story left this month. [Sign up for Medium and get an extra one](#)



Alex Teixeira

Follow

Dec 13, 2021 · 8 min read · ✨ · 🎧 Listen



Save



# Splunk IOC Scanner: a use case every-single-SOC needs



**TL;DR:** `tstats + term()` + `walklex` = super speedy (and accurate) queries. Leveraging Splunk terms by addressing a simple, yet highly demanded SecOps use case.

## How do you search for IOCs in Splunk?

When you have an IP address, do you map all data sources that might contain a valid IP address entry? What if a field is not CIM compliant, how do you add this into your scope? Out-of-the-box data models cannot help you here.

What if even after mapping all potential fields the query is still running super slow? This is the problem we are going to approach in this post.

---

*Security Operations teams will search for IOCs against Splunk events— using good or badly designed queries — you want it or not.*

*So you better provide analytics & interfaces for them to perform their job properly while avoiding performance issues and more importantly: false-negatives.*

---

I've been thinking about writing this one for a while and since the infamous Log4j vulnerability was published along with many IOC lists, I thought this would be a good time for it.

Splunk provides various methods for speeding up search results, from summary indexing, to metrics indexes, to accelerated data models (DMs). There are a few overlooked ones not even seasoned Splunkers know about...

Today we focus on a *query-level* technique.

## **Long term solution: Data Preparation**

Dealing with large scale datasets or indexes most times requires a tremendous work on data analysis, not different from the challenges our data science colleagues face in a daily basis. The fact is *data preparation* takes time.

Evaluating what's the best approach (summary index, accelerated DMs, etc) is super hard. You not only need to know how data is going to be consumed (use cases) but also what's the overall cost (maintenance, performance impact).

## **Speedy searches with TSTATS + TERM()**

Some use cases share similar target data. For instance, account brute-force detection and privileged access reports will tap into *authentication* telemetry.

Until those datasets are prepared or optimized, Splunk content engineers need to leverage other means to get the job done. Here's where query optimization comes in handy.

I'm not going into the details about *tstats* and *term()*, just assume those will basically enable you to run faster queries. The following read is going to provide you a good idea about them in case you are not familiar yet:

**Use CASE() and TERM() to match phrases**

Download topic as PDF If you want to search for a specific term or phrase in your Splunk index, use the CASE() or...

[docs.splunk.com](https://docs.splunk.com)

**Fun (or Less Agony) with Splunk Tstats | Deductiv**

Most of us have heard about how fast Splunk's tstats command can produce fast searches, but there's not much in the...

[www.deductiv.net](https://www.deductiv.net)

[Missing a great resource link for the community here? Please contact me!]

As a quick example, below is a query that will provide back as a result all *index* and *sourcetype* pairs containing the word (term) 'mimikatz':

```
| tstats count where index=* TERM(mimikatz) by index, sourcetype
```

You can try that with other terms. I even suggest a [simple exercise](#) for quickly discovering alert-like keywords in a new data source:

```
| tstats count where index=any TERM(exploit) by index, sourcetype,
_time span=1d | eval matches="exploit"
```

```
| append [ | tstats count where index=any TERM(malware) by index,
sourcetype, _time span=1d | eval matches="malware" ]
| append [ | tstats count where index=any TERM(virus) by index,
sourcetype, _time span=1d | eval matches="virus" ]
| append [ | tstats count where index=any TERM(attack) by index,
sourcetype, _time span=1d | eval matches="attack" ]
| append [ | tstats count where index=any TERM(brute) by index,
sourcetype, _time span=1d | eval matches="brute" ]
| append [ | tstats count where index=any TERM(alert) by index,
sourcetype, _time span=1d | eval matches="alert" ]
| append [ | tstats count where index=any TERM(violation) by index,
sourcetype, _time span=1d | eval matches="violation" ]
| append [ | tstats count where index=any TERM(critical) by index,
sourcetype, _time span=1d | eval matches="critical" ]
| append [ | tstats count where index=any TERM(detected) by index,
sourcetype, _time span=1d | eval matches="detected" ]
| append [ | tstats count where index=any TERM(botnet) by index,
```

Open in app ↗

Sign up

Sign In



machtes by index, sourcetype

So why not searching *everything* in Splunk using *tstats* and *term()*? Besides the significant change in terms of UX, there are more caveats.

The simple (but still complicated) answer is: not all *keyword* traces will make into a Splunk term containing only the searched keyword itself.

## Walklex: discover all terms

In the past, I've had the pleasure of working with many super talented folks while a member of Splunk PS, one of them is Richard Morgan.

His presentation at last .conf is a **must** for Splunk developers and analytics engineers: TSTATS and PREFIX. Why am I mentioning that?

After watching this awesome talk, I had an idea for updating the "IOC Scanner" dashboard I've been deploying to pretty much every single customer since 2017 while working as a contractor.



Nevertheless, I'm still not leveraging PREFIX() in a production use case -yet- but this will come handy at some point I'm pretty sure. What caught my attention was the use of walklex command to discover search *terms*.

After understanding how Splunk determines what is a term or not, it's easy to guess that not all values we pass along to TERM() directive will work.

Now, assuming vast majority of IOC values (IP, hash, etc) will not contain a major break (ex.: space), we basically need to discover how those values are saved as terms within Splunk as that fits a TERM() use case.

To perform this analysis, you basically run the following search query:

```
| walklex index=webproxy type=term
```

The output produces a *term* field  7 |  | values of terms observed within the period set in the query.

There are other values for the *type* parameter: *field*, *fieldvalue*. Each one has its own use cases as this is a really powerful tool for developers!

### How is an IPv4 address handled by Splunk?

It depends on how the raw log looks like.

Well, for most values, we will be able to search using TERM(IP) just like the 'Mimikatz' example seen earlier, however, the following raw log will have its terms broken down in a slightly different way:

```
[2021-11-11 18:22:16] WEBPROXY1 ClientIP=10.1.1.1 www.google.com
```

If you search for the website using TERM(www.google.com), you will definitely get a hit from that log example. However, if searching for the source IP address using similar technique TERM(10.1.1.1), **no results are seen!**

Why?!

Below are the actual terms from that particular log entry — after they are broken down by MAJOR breakers (space included):

```
2021-11-11
18:22:16
WEBPROXY1
ClientIP=10.1.1.1
www.google.com
```

As you might already guessed, the following search would yield a hit for that particular log record though:

```
| tstats count where index=webproxy TERM(ClientIP=10.1.1.1)
```

You actually cannot find the entry by using the IP address alone because after Splunk's MINOR breakers are applied (*dot* and *equal* signs included) here's how the relevant pieces end up in the TSIDX file:

```
...
ClientIP
10
1
...
```

## Building an accurate and fast IOC Scanner

I need to highlight *fast* is *relatively* speaking. What I mean by fast here is when compared to traditional methods (raw and even accelerated DM search):

```
index=webproxy ClientIP=10.1.1.1
| tstats summariesonly=1 count
  FROM datamodel=Web
  WHERE Web.src_ip=10.1.1.1
  ...
```

Besides faster *results*, this method is definitely faster to *build* since you don't even need to know which fields you actually need to search for (see below).

*This method applies to any data source, regardless if field extractions are properly done or not, so besides speed, that's super appealing in large scale environments!*

What I mean with (more) *accurate* is that you can also uncover gaps in your field extraction state by using this method. That is, if you miss the opportunity to extract a relevant value, by leveraging TERM() you can still find it.

In other words, chances of **false-negatives** are **minimized**.

Overall, for 24h, 7d or even 30d *lookback* periods the performance is staggering! Of course that also depends on how dense/large your index is, your setup (disk/IO) and how many terms are searched concurrently.

## A walk in the park

With the following *walklex* command you are able to discover most prevalent 'term prefixes' assuming you might have an equal sign (=) or a column (:) character (minor breakers) in between a field name and a value:

```
1 | walklex index=YOURINDEX type=term
2 | where match(term, "[^=:]+(=|:)[^=:]+$")
3 | eval termprefix=replace(term, "^[^=:]+(=|:)[^=:]+$", "\\1")
4 | stats count by termprefix
```

Now, if you run that *walklex* command against all your relevant indexes and you add the *index* to the stats command *group by* clause, you then have all the potential 'term prefixes' you need.

After shortlisting the relevant prefixes, you will be able to define the building block for a super fast search query, while dramatically reducing the chances of blind spots (false-negative):

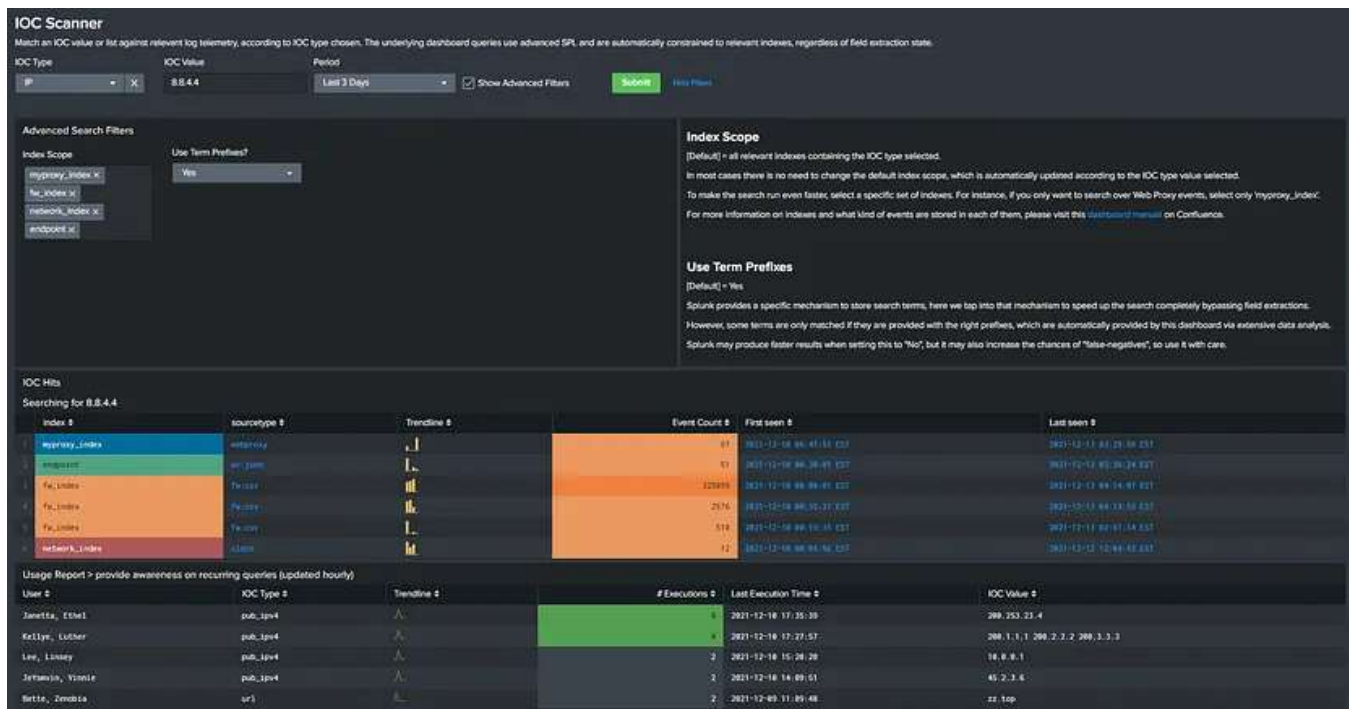
```
| tstats count AS hit_count
  WHERE index IN (<list of indexes>) AND (
    TERM(<iocvalue>) OR TERM(<iocprefix><iocvalue>)
  )
```

Note that there might be some tricky cases I am not going to explore here for keeping this simple. For instance, what if an IP address is only available in the log along with the destination port? Ex.: 10.1.1.1:8080 (term suffix).

## The dashboard

I'm not going into the details here, but once you master that method, building a dashboard is pretty straightforward. You basically build the query dynamically, according to the parameters set by the user.

Here's an example:



## How does it work?

Store each IOC type & index attributes in a lookup (index scope, prefixes) and load those according to the IOC type selected. Allow the user to narrow the search down to specific indexes or to toggle the usage of term prefixes.

The *drilldown* from the 'IOC Hits' panel can be customized for many things, from showing the raw events, to opening another analytic dashboard with more details. Use your creativity!

Handling a list or even a CSV file is not a big deal. There are many ways to build upon this concept, happy to bounce ideas off!

Kudos to [Myke Hamada](#) for helping me add the last bits to this one!

Have fun!

---

**Get an email whenever Alex Teixeira publishes.**

Your email

---



[Subscribe](#)

By signing up, you will create a Medium account if you don't already have one. Review our [Privacy Policy](#) for more information about our privacy practices.

[About](#) [Help](#) [Terms](#) [Privacy](#)

Get the Medium app

