


# Optimizing Splunk Knowledge Objects

Martin Müller

Professional Services Consultant

Consist Software Solutions GmbH

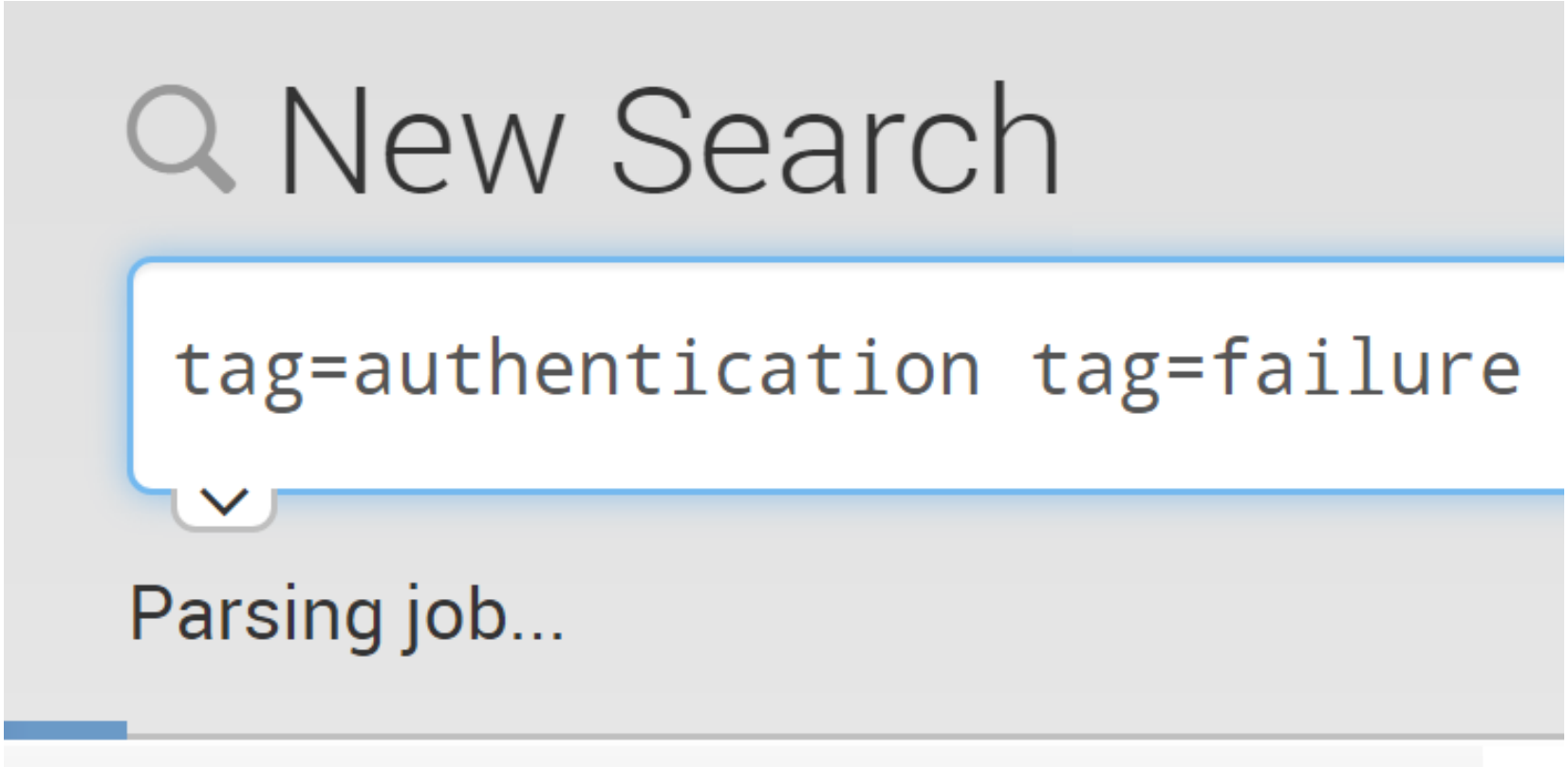
splunk>

# Disclaimer

During the course of this presentation, we may make forward looking statements regarding future events or the expected performance of the company. We caution you that such statements reflect our current expectations and estimates based on factors currently known to us and that actual events or results could differ materially. For important factors that may cause actual results to differ from those contained in our forward-looking statements, please review our filings with the SEC. The forward-looking statements made in the this presentation are being made as of the time and date of its live presentation. If reviewed after its live presentation, this presentation may not contain current or accurate information. We do not assume any obligation to update any forward looking statements we may make.

In addition, any information about our roadmap outlines our general product direction and is subject to change at any time without notice. It is for informational purposes only and shall not be incorporated into any contract or other commitment. Splunk undertakes no obligation either to develop the features or functionality described or to include any such feature or functionality in a future release.

# Why are we here?



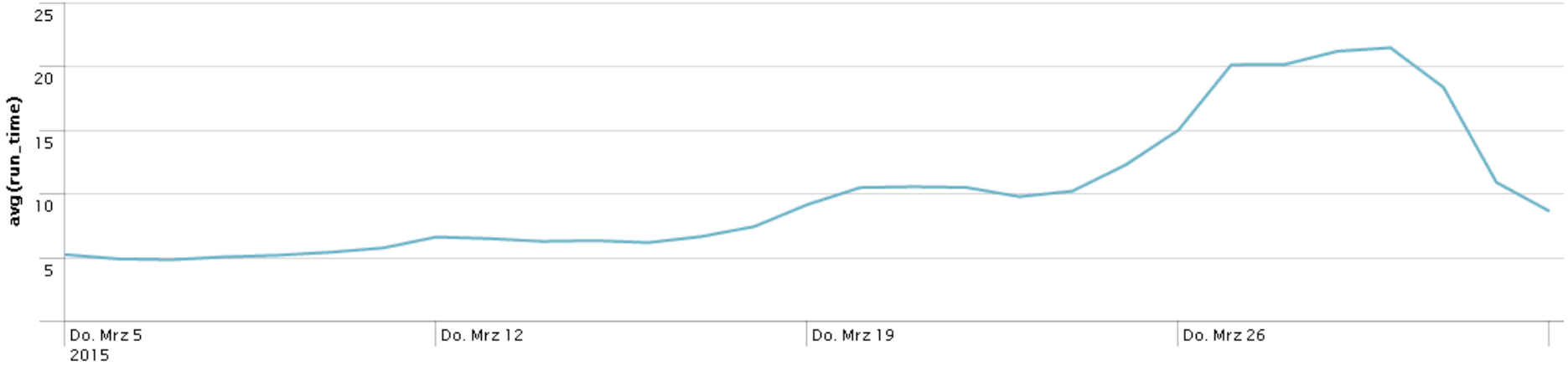
# Why are we here?

"Oversized litsearch is the largest performance problem we face in our environment."

*- Jacob Wilkins, General Electric*

# Why are we here?

- Observed search run time progression during development




- Massive growth in job startup time
- Knowledge Object optimization reduced that overhead by 80%

# Who‘s that guy?

- Professional Services Consultant, Certified Architect, Splunk-It-All
- Five years at EMEA Splunk Partner CONSIST Business Information Technology
- Heavy Splunker since 2012

- Get in touch with me: martin.mueller@consist.de
- Give karma at Splunk Answers: martin_mueller
- Hang in #splunk on Efnet: martin_m

# Session Objectives

- Understand how Splunk turns a search into results
- Learn how to recognize if you have a problem (Spoiler Alert: You do!)
- Use this to your advantage when specifying search-time knowledge

- Covered knowledge objects:
  - Fields
  - Reverse Lookups
  - Eventtypes
  - Tags


Let’s dive in…

splunk>

# …but first, to the Job Inspector!

- normalizedSearch: Ultra-verbose stage of search assembly

| normalizedSearch | litsearch index=_audit ( action=search OR ( sourcetype=audittrail ) ) \| litsearch index=_audit action=search \| fields keepcolorder=t "*" "_bkt" "_cd" "_si" "host" "index" "linecount" "source" "sourcetype" "splunk_server" |
|---|---|

- Performance stats, e.g. time spent assembling the normalizedSearch


- Links to search.log to look for more hidden performance hogs
- More at http://docs.splunk.com/Documentation/Splunk/latest/ Knowledge/ ViewsearchjobpropertieswiththeJobInspector


# Fields

# Calculated Fields (1)

- TA-splunk, props.conf: `[audittrail]`
  `EVAL-action=case(condN, valN, 1=1, action)`
- Splunk‘s assumption about looking for indexed tokens doesn‘t hold
- No way to translate the eval expression into tokens
- Plain Search: `index=_audit action=search`
  normalizedSearch: `index=_audit (action=search OR (sourcetype=audittrail))`
- Load all events for that stanza plus events with the token, filter later

# Calculated Fields (2)

- What if you're not searching for that sourcetype?
- ```
  index=_internal sourcetype=splunk*
    action=logout
  index=_internal sourcetype="splunk*"
   (action=logout OR (sourcetype=audittrail))
  ```
- Splunk expands each segment of your search on its own
- For each calculated field, add stanza to every search for that field
- This is only the beginning of normalizedSearch overhead!
  ```

# Field Aliases

- Sourcetype A has field `username`, sourcetype B has field `uid`, …
- Field aliases can normalize this to `user` over all sourcetypes
- `sourcetype=A user=martin` yields this normalizedSearch:

```
sourcetype=A (
  ((sourcetype=A) AND ((username=martin))) OR
  ((sourcetype=B) AND ((uid=martin))) OR
  ((sourcetype=audittrail) AND ((uid=martin)))
) OR (user=martin)
```

- All field aliases for all sourcetypes are used in all searches!

# A real-world example

- Splunk App for Enterprise Security 3.3.1
- The TAs shipped define 19 field aliases for `user`
- Your environment will have additional TAs
- Watch your normalizedSearch strings and search startup time grow

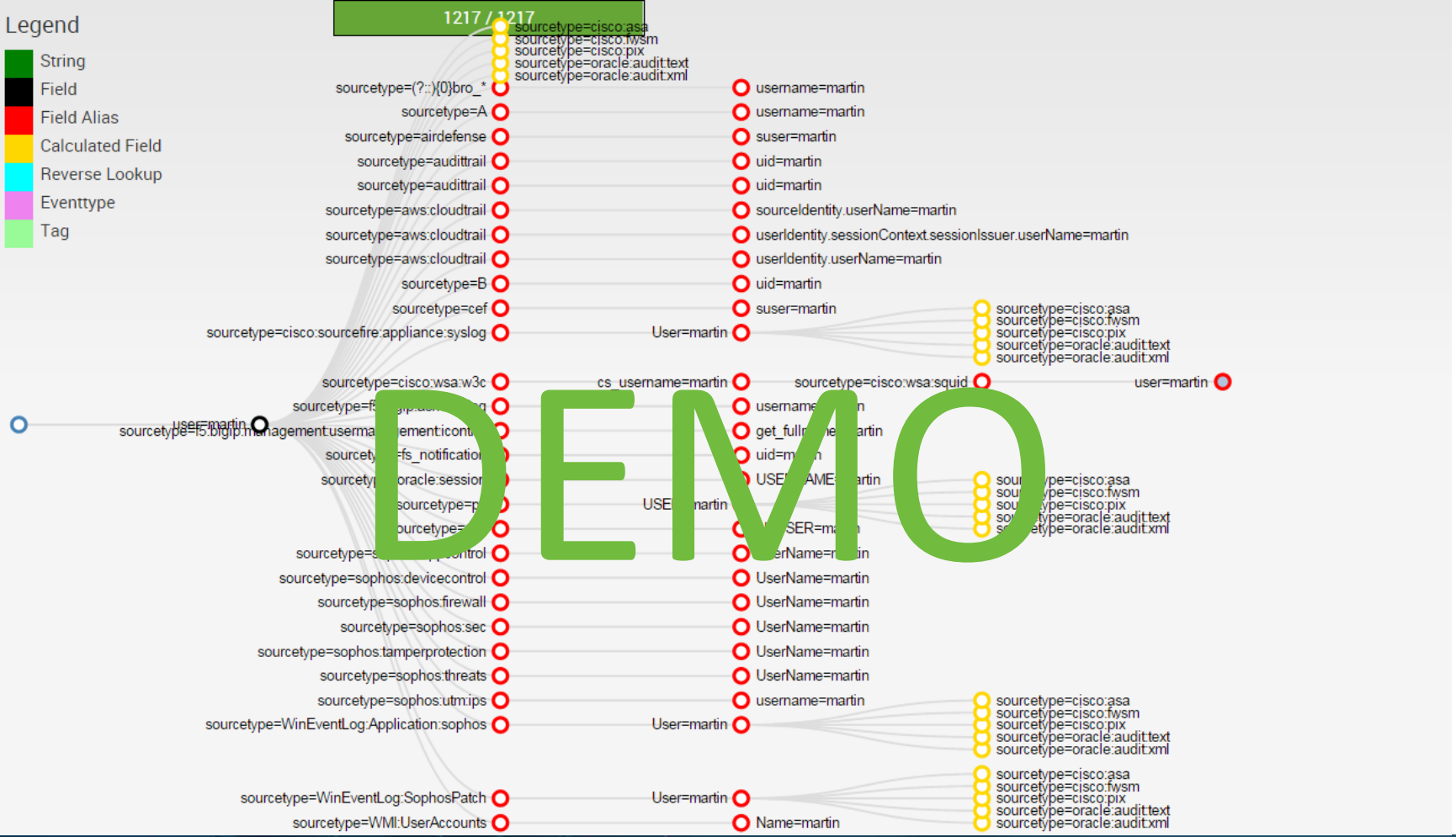
- Let‘s not forget the upside though: Without standardized field names, searching over different sourcetypes would be impossible
- Are you building a TA? Extract standardized field names directly!

# A real-world example

- Searching for `user=martin` yields 2kB of `normalizedSearch`:

```
((((sourcetype="*") AND ((username=martin))) OR ((sourcetype=A) AND ((username=martin))) OR
((sourcetype=B) AND ((uid=martin))) OR ((sourcetype="WMI:UserAccounts") AND ((Name=martin)))
OR ((sourcetype="WinEventLog:Application:sophos") AND ((User=martin))) OR
((sourcetype="WinEventLog:SophosPatch") AND ((User=martin))) OR ((sourcetype="WinEventLog:SophosDefense") AND
((suser=martin))) OR ((sourcetype=audittrail) AND ((uid=martin))) OR
((sourcetype="aws:cloudtrail") AND (("sourceIdentity.userName"=martin) OR
("userIdentity.sessionContext.sessionIssuer.userName"=martin) OR
("userIdentity.userName"=martin))) OR ((sourcetype=cef) AND ((suser=martin)) OR
((sourcetype="cisco:sourcefire:appliance:syslog") AND (User=martin))) OR
((sourcetype="cisco:wsa:w3c") AND ((cs_username=martin))) OR
((sourcetype="f5:bigip:asm:syslog") AND (username=martin))) OR
((sourcetype="f5:bigip:management:usermanagement:control") AND ((gt_fullname=martin))) OR
((sourcetype=fs_notification) AND ((uid=martin))) OR ((sourcetype="oracle:session") AND
((USERNAME=martin)) OR ((sourcetype=sav) AND ((USER=martin))) OR ((sourcetype=sav) AND
((LI_USER=martin))) OR ((sourcetype="sophos:appcontrol") AND ((UserName=martin))) OR
((sourcetype="sophos:devicecontrol") AND ((UserName=martin))) OR
((sourcetype="sophos:firewall") AND ((UserName=martin))) OR ((sourcetype="sophos:sec") AND
((UserName=martin))) OR ((sourcetype="sophos:tamperprotection") AND ((UserName=martin))) OR
((sourcetype="sophos:threat") AND ((UserName=martin))) OR ((sourcetype="sophos:utm:ips") AND
((username=martin))) OR (user=martin) OR (sourcetype="cisco:asa") OR
(sourcetype="cisco:fwsm") OR (sourcetype="cisco:pix") OR (sourcetype="oracle:audit:text") OR
(sourcetype="oracle:audit:xml"))
```

NOT PRETTY!


Legend
String
Field
Field Alias
Calculated Field
Reverse Lookup
Eventtype
Tag
1217 / 1217
sourcetype=cisco:asa
sourcetype=cisco:fwsm
sourcetype=cisco:pix
sourcetype=oracle:audit:text
sourcetype=oracle:audit:xml
sourcetype=(?::){0}bro_*
sourcetype=A
sourcetype=airdefense
sourcetype=audittrail
sourcetype=audittrail
sourcetype=aws:cloudtrail
sourcetype=aws:cloudtrail
sourcetype=aws:cloudtrail
sourcetype=B
sourcetype=cef
sourcetype=cisco:sourcefire:appliance:syslog
sourcetype=cisco:wsa:w3c
user=martin
sourcetype=f5:bigip:management:usermanagement:icont
sourcetype=fs_notification
sourcetype=oracle:session
sourcetype=sophos:devicecontrol
sourcetype=sophos:firewall
sourcetype=sophos:sec
sourcetype=sophos:tamperprotection
sourcetype=sophos:threats
sourcetype=sophos:utm:ips
sourcetype=WinEventLog:Application:sophos
sourcetype=WinEventLog:SophosPatch
sourcetype=WMI:UserAccounts
username=martin
username=martin
suser=martin
uid=martin
uid=martin
sourceIdentity.userName=martin
userIdentity.sessionContext.sessionIssuer.userName=martin
userIdentity.userName=martin
uid=martin
suser=martin
User=martin
cs_username=martin
username=martin
get_full
uid=m
USE
USE martin
UserName=martin
UserName=martin
UserName=martin
UserName=martin
UserName=martin
username=martin
User=martin
User=martin
Name=martin
sourcetype=cisco:asa
sourcetype=cisco:fwsm
sourcetype=cisco:pix
sourcetype=oracle:audit:text
sourcetype=oracle:audit:xml
sourcetype=cisco:wsa:squid
user=martin
DEMO

# Fields Recap

- Each search segment is expanded on its own without context
- props.conf for one sourcetype will radiate into normalizedSearch of other sourcetypes when field names match

- Avoid calculated fields and field aliases entirely where possible
  - Extract fields using standardized names in the first place!
  - Some calculated fields can be replaced with lookups
- Monitor their effects where unavoidable
- Both are fine for fields you only use as output


# Reverse Lookups

splunk>

# How reverse lookups work

- Automatic lookup in props.conf: `[splunk_web_access]`
  `LOOKUP-ul = user_location user OUTPUT location`
- Reverse lookup: Search for `location` rather than `user`:
  `index=_internal location="Las Vegas"`
- Splunk translates that into this normalizedSearch:

```
index=_internal
(((sourcetype=splunk_web_access) AND
  ((user=Martin) OR (user=Tom))
)) OR (location="Las Vegas")
```

# Actually, I lied…

```
index=_internal (((sourcetype=splunk_web_access) AND
((((((sourcetype=A) AND ((username=Martin))) OR
    ((sourcetype=B) AND ((uid=Martin))) OR
    ((sourcetype=audittrail) AND ((uid=Martin)))) OR
(user=Martin))) OR
 ((((((sourcetype=A) AND ((username=Tom))) OR
    ((sourcetype=B) AND ((uid=Tom))) OR
    ((sourcetype=audittrail) AND ((uid=Tom)))) OR
(user=Tom)))))) OR (location="Las Vegas")
```

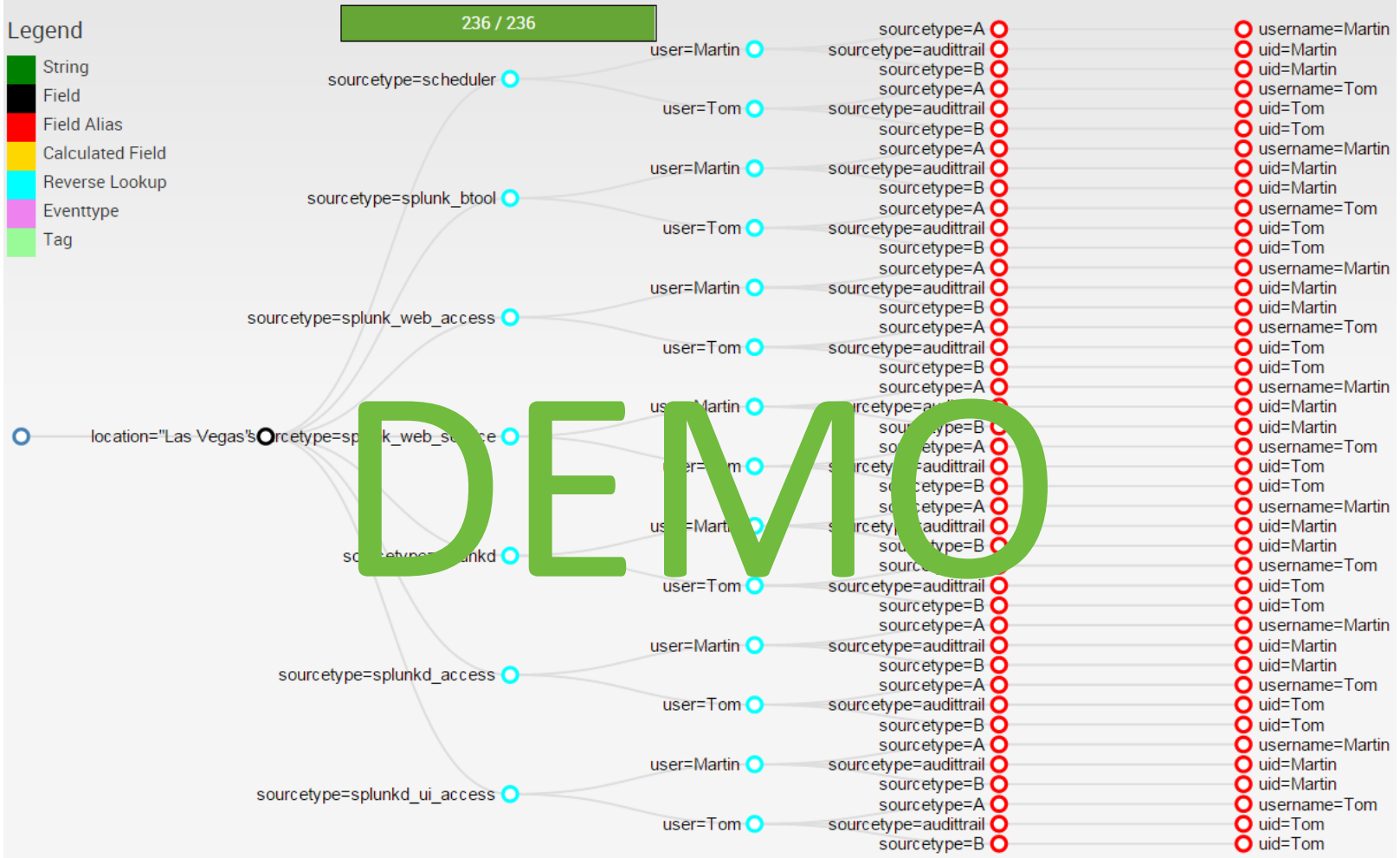
- Despite defining the lookup on `splunk_web_access`, other sourcetypes' props.conf settings radiate into this search

# Expanding to more sourcetypes

- Splunk's _internal index has seven sourcetypes with a `user` field

```
index=_internal (((sourcetype=scheduler) AND ((((((sourcetype=A) AND ((username=Martin))) OR ((sourcetype=B) AND
((uid=Martin))) OR ((sourcetype=audittrail) AND ((uid=Martin)))) OR (user=Martin))) OR (((((sourcetype=A) AND
((username=Tom))) OR ((sourcetype=B) AND ((uid=Tom))) OR ((sourcetype=audittrail) AND ((uid=Tom)))) OR (user=Tom)))))
OR ((sourcetype=splunk_btool) AND ((((((sourcetype=A) AND ((username=Martin))) OR ((sourcetype=B) AND ((uid=Martin))))
OR ((sourcetype=audittrail) AND ((uid=Martin)))) OR (user=Martin))) OR (((((sourcetype=A) AND ((username=Tom))) OR
((sourcetype=B) AND ((uid=Tom))) OR ((sourcetype=audittrail) AND ((uid=Tom)))) OR (user=Tom))))) OR
((sourcetype=splunk_web_access) AND ((((((sourcetype=A) AND ((username=Martin))) OR ((sourcetype=B) AND
((uid=Martin))) OR ((sourcetype=audittrail) AND ((uid=Martin)))) OR (user=Martin))) OR (((((sourcetype=A) AND
((username=Tom))) OR ((sourcetype=B) AND ((uid=Tom))) OR ((sourcetype=audittrail) AND ((uid=Tom)))) OR (user=Tom)))))
OR ((sourcetype=splunk_web_service) AND ((((((sourcetype=A) AND ((username=Martin))) OR ((sourcetype=B) AND
((uid=Martin))) OR ((sourcetype=audittrail) AND ((uid=Martin)))) OR (user=Martin))) OR (((((sourcetype=A) AND
((username=Tom))) OR ((sourcetype=B) AND ((uid=Tom))) OR ((sourcetype=audittrail) AND ((uid=Tom)))) OR (user=Tom)))))
OR ((sourcetype=splunkd) AND ((((((sourcetype=A) AND ((username=Martin))) OR ((sourcetype=B) AND ((uid=Martin))) OR
((sourcetype=audittrail) AND ((uid=Martin)))) OR (user=Martin))) OR (((((sourcetype=A) AND ((username=Tom))) OR
((sourcetype=B) AND ((uid=Tom))) OR ((sourcetype=audittrail) AND ((uid=Tom)))) OR (user=Tom))))) OR
((sourcetype=splunkd_access) AND ((((((sourcetype=A) AND ((username=Martin))) OR ((sourcetype=B) AND ((uid=Martin)))
OR ((sourcetype=audittrail) AND ((uid=Martin)))) OR (user=Martin))) OR (((((sourcetype=A) AND ((username=Tom))) OR
((sourcetype=B) AND ((uid=Tom))) OR ((sourcetype=audittrail) AND ((uid=Tom)))) OR (user=Tom))))) OR
((sourcetype=splunk_ui_access) AND ((((((sourcetype=A) AND ((username=Martin))) OR ((sourcetype=B) AND
((uid=Martin))) OR ((sourcetype=audittrail) AND ((uid=Martin)))) OR (user=Martin))) OR (((((sourcetype=A) AND
((username=Tom))) OR ((sourcetype=B) AND ((uid=Tom))) OR ((sourcetype=audittrail) AND ((uid=Tom)))) OR
(user=Tom)))))) OR (location="Las Vegas")
```

NOT PRETTY!


Legend
String
Field
Field Alias
Calculated Field
Reverse Lookup
Eventtype
Tag
236 / 236
location="Las Vegas"
sourcetype=scheduler
sourcetype=splunk_btool
sourcetype=splunk_web_access
sourcetype=splunkd_access
sourcetype=splunkd_ui_access
user=Martin
user=Tom
sourcetype=A
sourcetype=audittrail
sourcetype=B
username=Martin
username=Tom
uid=Martin
uid=Tom
DEMO

# A location with more than two users?

- 50 users produce a 72kB normalizedSearch that broke PowerPoint
- Noticeable overhead during `Parsing Job...` phase


- That‘s with three field aliases and no calculated fields – imagine 20+!

- Above 50 values per lookup Splunk will revert to „classic“behavior: Load all events, filter later

# Mitigation strategies (1)

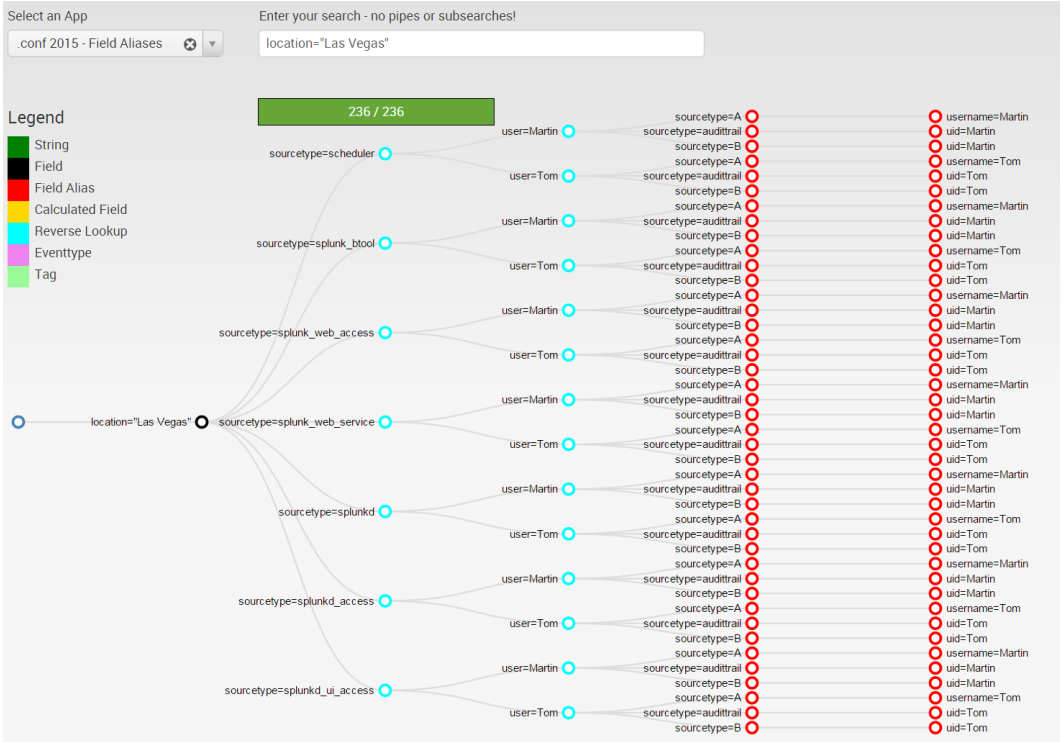
- Subsearch using inputlookup
  index=_internal [inputlookup user_location | search location="Las Vegas" | fields user]
- ✅ Removes the per-sourcetype duplication
- ✅ Lets you choose between reverse lookups and *classic* behavior
- ⚠ Ignores the configured knowledge per sourcetype
- ⚠ More effort required to write and maintain searches
- ⚠ Not eventtype-compatible
- ⚠ Subsearch overhead

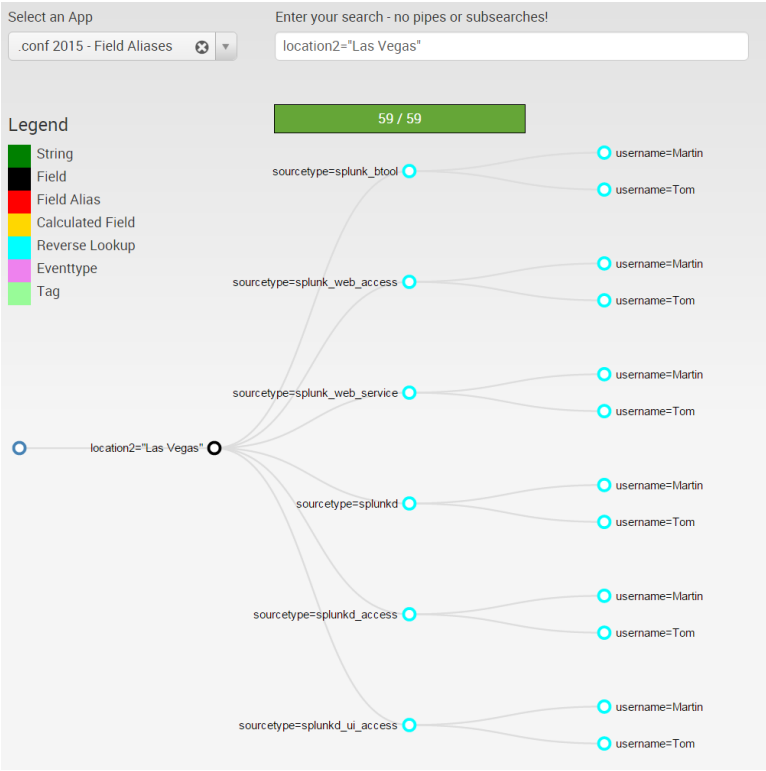
# Mitigation strategies (2)

- Define the per-sourcetype automatic lookup using sourcetype-specific **input** fields

```
LOOKUP-ul = user_location user AS username
OUTPUT location
```

- ✓ Removes the per-alias duplication
- ✓ Transparent to the search and user
- ⚠ More effort required to write and maintain knowledge objects
- ⚠ Retains the per-sourcetype duplication




Removed 80% of key-value pairs from the normalizedSearch!

# Mitigation strategies (3)

- Define the per-sourcetype automatic lookup using sourcetype-specific **output** fields

```
LOOKUP-ul = user_location user OUTPUT location
AS sourcetype_location
```

- ✓ Removes the per-sourcetype duplication
- ⚠ Not transparent at all
- ⚠ More effort required to write and maintain knowledge objects
- ⚠ Only really viable if hidden behind eventtypes and/or tags
- ⚠ Retains the per-alias duplication

# Mitigation strategies (4)

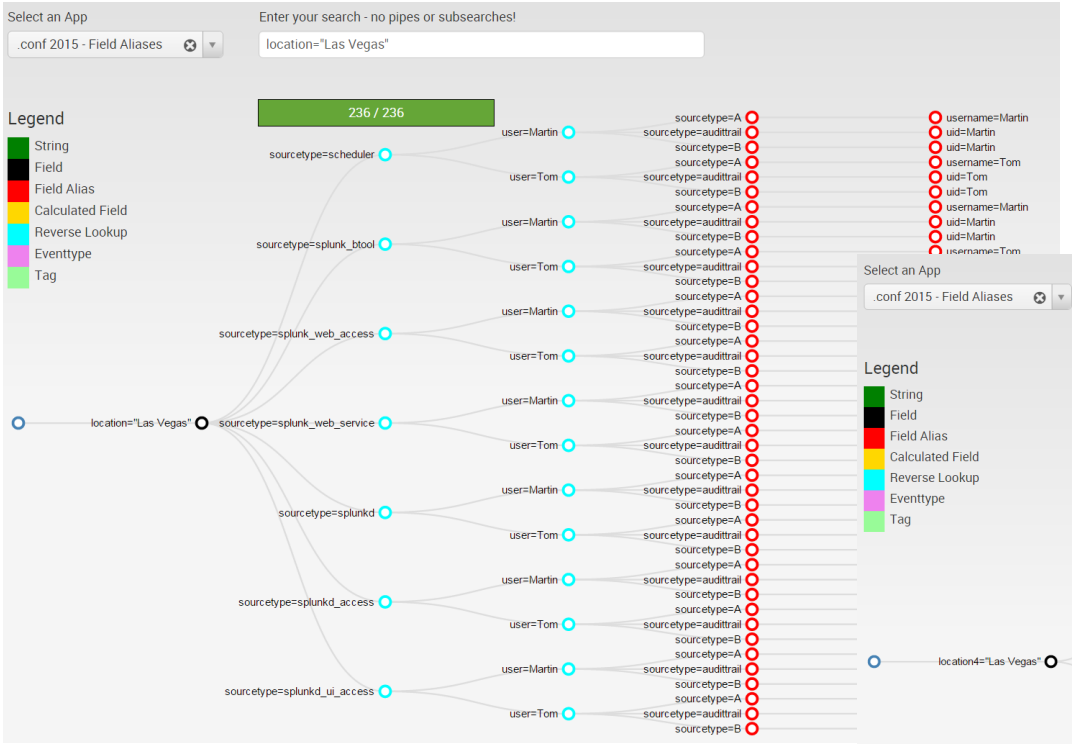
- Replace per-sourcetype lookups with broader props.conf stanzas
- Wildcards on source or host
  `[source::*access.log*]`
- Unofficial: Wildcards on sourcetype
  `[(?::){0}splunk*]`

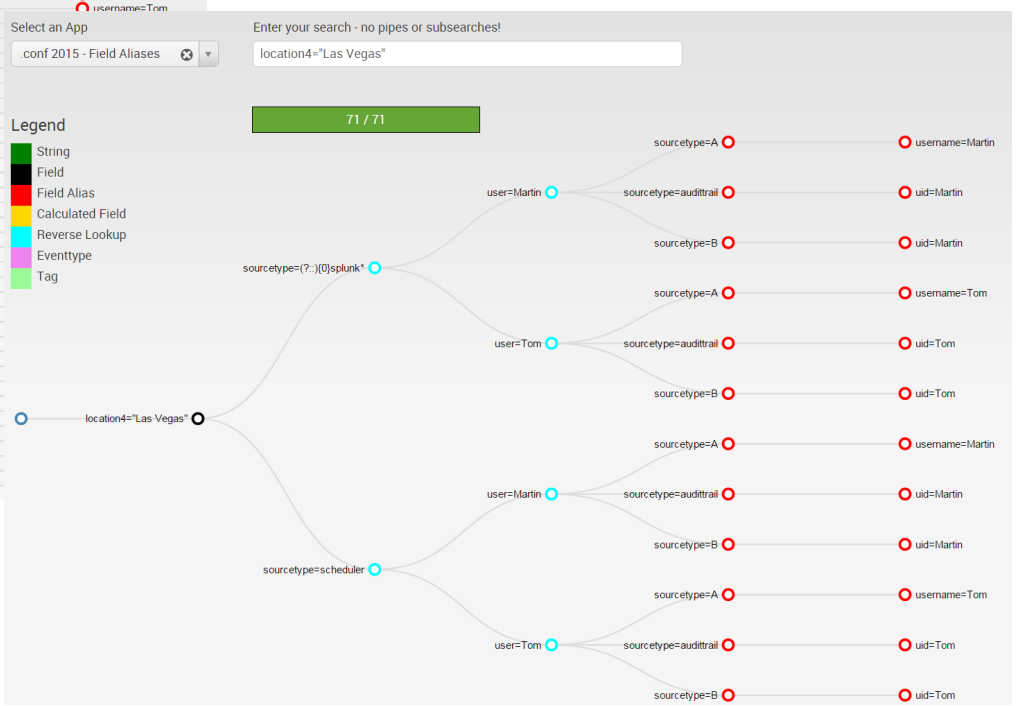
✓ Removes the per-sourcetype duplication

✓ Transparent to the search and user

⚠ Sourcetype wildcards are neither documented nor supported

⚠ Retains the per-alias duplication




70% key-value pair reduction!

# Indexed tokens footnote

- The normalizedSearch generated by reverse lookups can be efficient:

```
index=_internal location="Las Vegas"
index=_internal
(((sourcetype=splunk_web_access) AND
  ((user=Martin) OR (user=Tom))
)) OR (location="Las Vegas")
```

- But: Splunk is looking for a literal `location="Las Vegas"`!
- Watch out for `location=0` or similar values that aren‘t unique-ish
- This can blow up your scanCount and search duration
- More on dealing with indexed tokens after the end of the deck


# Eventtypes

# How eventtypes work

- Store a search filter or fragments thereof in a reusable box
- No pipes, no subsearches
- Run search and see `searchCanBeEventType` in Job Inspector
- `eventtype=foo` expands to the stored search fragment
- `eventtype=f*` expands to an `OR`'d list of matching eventtypes
- Events that match an eventtype have their `eventtype` field set, regardless of whether the eventtype was used in the search or not

# What are eventtypes good at?

- Two different systems likely don‘t log login attempts the same way
- Define eventtypes for each system, search on eventtypes
  - Tag your eventtypes and search on tags
- Configured knowledge simplifies searches
- Great way to hide complexity from the searcher
- Add systems to existing searches without touching searches
- Even when not searching on eventtypes, looking at the `eventtype` field helps quickly understand results

# Splunk login example

- TA-splunk, eventtypes.conf: `[splunk_access]`
  `search = index=_audit "action=login attempt" NOT "action=search"`
  normalizedSearch: `((index=_audit "action=login attempt" NOT "action=search"))`
- Note how Splunk chose not to use `action="login attempt"`!
- Avoids the wrath of calculated fields and aliases in the search
- Search relies on structure of raw events instead of field extractions
- The results contain the CIM-compatible `action` regardless


# Tags

# How tags work

- Give a set of `field=value` pairs a common name
- No wildcarded `field=v*`
  - can be worked around with tagged eventtypes
- `tag=foo` expands to the list of `field=value` pairs individually
- `tag=f*` expands to an `OR`'d list of matching tags
- Events that match a tag have their `tag` field set accordingly
- For each tagged `field`, additionally set `tag::field` field

# What are tags good at?

- Homogenize system-specific values to allow unified searches
- Great in combination with eventtypes:
  - Eventtypes define system-specific searches
  - Tags on those eventtypes provide a common interface
  - Searches on those tags don‘t need to know the systems particularly well
- Also great in combination with normalized field names and values
  - The unified searches find events over many systems
  - The returned results also provide homogenous data back to you
- That‘s the Splunk Common Information Model in a nutshell
- Further reading at http://docs.splunk.com/Documentation/CIM

# Splunk login example

- TA-splunk, tags.conf: `[eventtype=splunk_access]`
  `application = enabled`
  `authentication = enabled`
- The search `tag=application tag=authentication` yields
  `(((index=_audit "action=login attempt" NOT "action=search"))) (((index=_audit "action=login attempt" NOT "action=search")))`
- The eventtype is included twice!

# How tags really work

- Search for `tag=application tag=authentication`
- Splunk won‘t look for `field=value` pairs matching both tags
- Splunk will treat the search like this:
  `(tag=application) (tag=authentication)`
- Each tag is expanded individually
- `field=value` pairs will be included once per matching tag
- This can lead to even larger normalizedSearch strings!

# A real-world example

- Splunk_TA_Oracle defines a handful of tagged eventtypes
- Four match `tag=database tag=instance tag=stats`
- Expanding each tag on its own yields sixteen eventtypes!
- Every TA is influenced by every other TA: „Tag Expansion Explosion“

# Mitigation Strategies

- Avoid long lists of tags mapping to the same `field=value`
  - Especially with eventtypes and reverse lookups
- Use distributive properties to reduce tag-eventtype redundancy
  - Instead of tagging every Splunk eventtype with `application`, consider tagging `sourcetype`, `host`, etc. with `application`
  - Instead of tagging special eventtypes for admin users with `privileged`, consider tagging those users or a reverse lookup field identifying them
- Look for what actually defines the tag in the real world
- Charm Splunk into optimizing how tags are expanded ☺


# Wrapping up

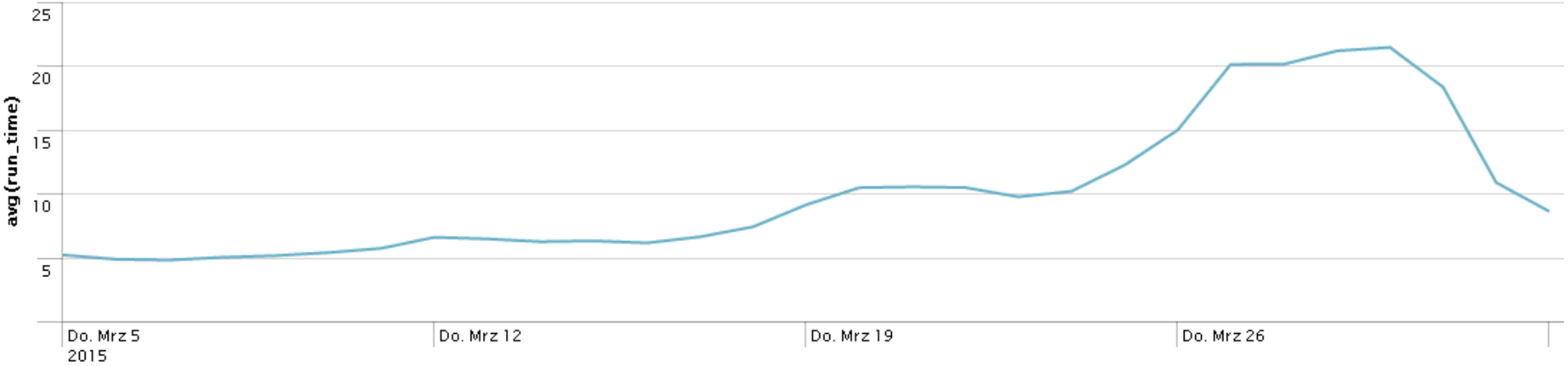
splunk>

# Dos and Don‘ts

- ⚠ Don‘t stop using field aliases, calculated fields, reverse lookups, etc.
- ⚠ Don‘t compromise maintainability for small gains

- ✔ Do take a good look at your environment
- ✔ Do identify and improve real performance hogs
- ✔ Do scope knowledge object sharing as narrowly as possible
- ✔ Do clean up unused knowledge objects and TAs
- ✔ Do keep monitoring as your knowledge object world grows

# Q&A


avg(run_time)
25
20
15
10
5
Do. Mrz 5
2015
Do. Mrz 12
Do. Mrz 19
Do. Mrz 26

# What Now?

Related breakout sessions and activities…

- You have access to your Splunk at .conf? Talk to me for a quick look!
- Grab the app: https://splunkbase.splunk.com/app/2871

- Duane & George: Beyond the Lookup Glass (Tuesday)
- Amrit & Jag: How splunkd Works (Tuesday)
- Duncan & Julian: Search Efficiency Optimization (Tuesday)
- Niklas: How to use CIM to Gain Security Awareness (Wednesday)
- Dritan: Notes on Optimizing Splunk Performance (later today!)


# THANK YOU

splunk>


# Fields: Optimizations Beyond Litsearch

splunk>

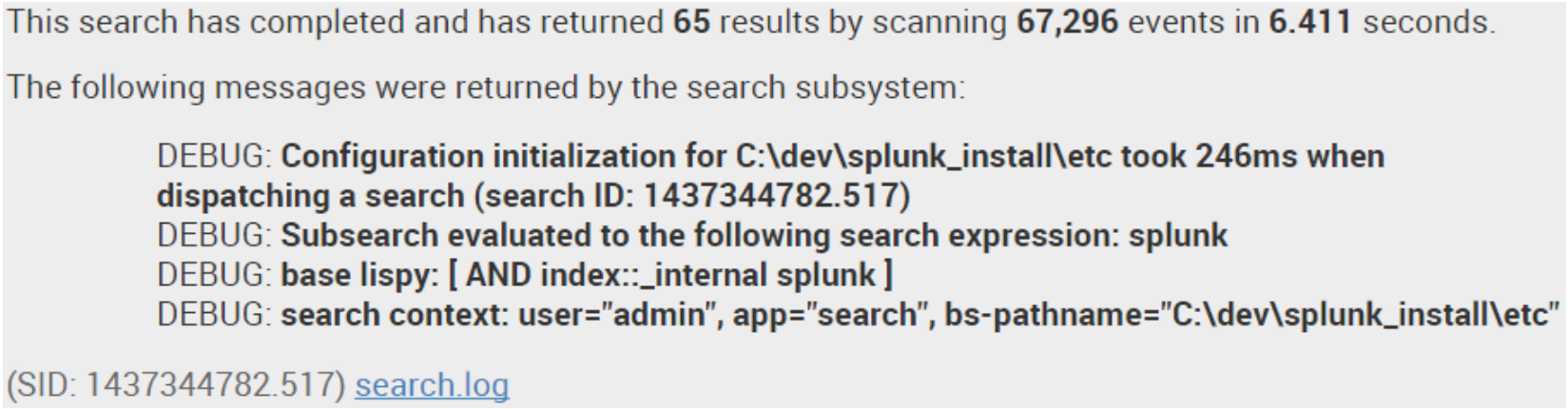
# Fields

"Let all values be indexed tokens, for indexed tokens power fast searches."

*- Splunk, late 2000s*

# Job Inspector continued

- base lispy: How did Splunk crawl its index for events?
- eventCount / scanCount: How efficient was the lispy-induced crawl?




- limits.conf: `[search_info] infocsv_log_level=DEBUG`

# How Splunk searches for field values (1)

```
index=_internal group=tpool
```

- Assume a field value is present as indexed tokens
- Load events containing those indexed tokens anywhere

```
[ AND index::_internal tpool ]
```

- Apply field extractions and filter again

```
07-21-2015 22:42:52.662 +0200 INFO Metrics -
group=tpool, name=indexertpool, qsize=0, ...
```

- Job Inspector: scanCount ≈ eventCount

# How Splunk searches for field values (2)

```
index=_internal qsize=0
[ AND index::_internal 0 ]
```

- Splunk returns the same event, but takes ages!

```
07-21-2015 22:42:52.662 +0200 INFO Metrics -
group=tpool, name=indexertpool, qsize=0, ...
```

| | |
|---|---|
| eventCount | 18225 |
| scanCount | 509781 |

- Default assumption works great iff field values are unique-ish

# Key-Value Tricks (1)

```
index=_internal qsize qsize=0
[ AND index::_internal qsize 0 ]
```

- Take advantage of default key-value field extractions

```
07-21-2015 22:42:52.662 +0200 INFO Metrics -
group=tpool, name=indexertpool, qsize=0, ...
```

| | |
|---|---|
| eventCount | 18225 |
| scanCount | 18691 |

- Flexible, zero-config speed-up that requires smart searchers!

# Key-Value Tricks (2)

- Move inline optimization to fields.conf

```
[qsize]
INDEXED_VALUE=[AND qsize <VALUE>]
```

- Adds the extra token `qsize`, whether the searcher likes it or not

```
index=_internal qsize=0
[ AND index::_internal qsize 0 ]
```

- fields.conf applies to **all** fields of that name, regardless of sourcetype
- This can break for multi-token values!

# Key-Value Tricks (3)

- Take it further and assemble longer tokens

```
[qsize]
INDEXED_VALUE=qsize=<VALUE>
```

- Rule out events with `qsize!=0` that contain a `0` elsewhere

```
index=_internal qsize=0
[ AND index::_internal qsize=0 ]
```

- This will even break for events with `qsize="0"` (major breaker)
- Be sure you know your data before fiddling with fields.conf!

# Wildcards (1)

- Splunk will only use indexed tokens for prefixes of wildcarded values
- `index=_internal component=`**`BucketMove*`**
  `[ AND index::_internal `**`bucketmove*`**` ]`
- `index=_internal component=*ucketMover`
  `[ AND index::_internal ]`
- Oops!

`07-21-2015 22:41:22.999 +0200 INFO `**`BucketMover`**` - idx=main Moving bucket=…`

# Wildcards (2)

- Force Splunk to use indexed tokens
- `index=_internal component=TERM(*ucketMover) [ AND index::_internal **\*ucketmover** ]`
- Much faster than loading all events, but there's a penalty for crawling the index without a prefix!


- fields.conf to remove the `TERM()` from all searches

```
[component]
INDEXED_VALUE=<VALUE>
```

# Fields Recap (Part 2)

- Indexed tokens are king
- scanCount performance hit when indexed tokens can‘t be used
- fields.conf optimizations can fix performance, but can break results