Forward-Looking Statements

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 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.

Splunk, Splunk>, Listen to Your Data, The Engine for Machine Data, Splunk Cloud, Splunk Light and SPL are trademarks and registered trademarks of Splunk Inc. in the United States and other countries. All other brand names, product names, or trademarks belong to their respective owners. © 2017 Splunk Inc. All rights reserved.
Me

- Integration Developer with Aplura, LLC
- Working with Splunk for ~8 years
- Written many Public Splunk Apps (on splunkbase.splunk.com)
- Current Member of the SplunkTrust
- Wrote the “Splunk Developer’s Guide” - introduction to Splunk App Development
- Active on #splunk on IRC, answers.splunk.com, and Slack
- Co-leader of Baltimore Usergroup
  - My Handle is “alacercogitatus” or just “alacer”
• Splunk
  • Admin
  • User
  • Architect
  • Evangelist
  • Sales Engineer
  • Anybody

• Want to learn about new search commands
• Enjoy Piña Coladas, getting caught in the rain (well maybe not)
• Intermediate experience with SPL (know how to “stats”)

Goals

• Show/expose you to possibly new commands
• Won’t become “expert” on these commands
• Take actionable items back to your business to “try new things”
Agenda

• Administrative/Generating

• Iterative

• Statistics

• SPLacks

NOTE: These commands have been verified for Splunk 7.0. Any other version may or may not work correctly.
Administrative (generating) Commands

- rest, makeresults, gentimes, metasearch, metadata, untable

- Commands that do not pull full (raw) events or are used to interact with Splunk
  - Provide information about data or the Splunk server(s)
  - These are all “generating” commands
The `rest` command reads a Splunk REST API endpoint and returns the resource data as a search result.¹

• MUST be the first search command in a search block
• Is “time agnostic” - It only queries - so time is not a factor in execution
• Limits results to what the requesting user is allowed to access

```
| rest /services/data/indexes splunk_server=local count=0
| dedup title
| fields title
```

---

This endpoint pulls the currently logged in user’s authorization context. With this information, you could restrict specific dashboards or searches by role, even more granular than the pre-built permissions.

<table>
<thead>
<tr>
<th>defaultApp</th>
<th>username</th>
<th>roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>launcher</td>
<td>ksmith</td>
<td>admin, windows-admin, winfra-admin</td>
</tr>
<tr>
<td>launcher</td>
<td>splunk-system-user</td>
<td>admin, splunk-system-role</td>
</tr>
</tbody>
</table>
makeresults

Generates the specified number of search results. If you do not specify any of the optional arguments, this command runs on the local machine and generates one result with only the _time field.

- New in 6.3
- Easy way to “spoof” data to experiment with evals, and other SPL commands
- Fast, lightweight
- Use it to restrict a search using it in a subsearch

```bash
index=_internal _indextime >
| makeresults
| eval it=now()-60
| return $it]
```

gentimes

Generates timestamp results starting with the exact time specified as start time. Each result describes an adjacent, non-overlapping time range as indicated by the increment value. This terminates when enough results are generated to pass the endtime value.¹

- Useful for generating time buckets not present due to lack of events within those time buckets
- Must be the first command of a search (useful with map, or append)
- “Supporting Search” - no real use case for basic searching

```
| gentimes start=10/1/15 end=10/5/15 |
```

<table>
<thead>
<tr>
<th>endhuman</th>
<th>endtime</th>
<th>starthuman</th>
<th>starttime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thu Oct 1 23:59:59 2015</td>
<td>1443758399</td>
<td>Thu Oct 1 00:00:00 2015</td>
<td>1443672000</td>
</tr>
<tr>
<td>Fri Oct 2 23:59:59 2015</td>
<td>1443844799</td>
<td>Fri Oct 2 00:00:00 2015</td>
<td>1443768400</td>
</tr>
<tr>
<td>Sat Oct 3 23:59:59 2015</td>
<td>1443931199</td>
<td>Sat Oct 3 00:00:00 2015</td>
<td>1443844800</td>
</tr>
<tr>
<td>Sun Oct 4 23:59:59 2015</td>
<td>1444017599</td>
<td>Sun Oct 4 00:00:00 2015</td>
<td>1443931200</td>
</tr>
</tbody>
</table>

¹ https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Gentimes
gentimes

```bash
| gentimes start=-9 end=-7 increment=1h
| map maxsearches=48 search="search earliest=$starttime$ latest=$endtime$ eventtype=splunkcraft2016 | bucket _time span=1h | top useother=f limit=5
  block_type by _time | fields - percent"
| timechart span=1h sum(count) by block_type useother=f
```
metasearch

Retrieves event metadata from indexes based on terms in the <logical-expression>. Metadata fields include source, sourcetype, host, _time, index, and splunk_server. 

• Useful for determining what is located in the indexes, based on raw data
• Does NOT present raw data
• Can only search on raw data, no extracted fields
• Can be tabled based on the metadata present
• Respects the time picker and default searched indexes

|metasearch| eventtype=splunkcraft2016|earliest=-24h@h|
metadata

The metadata command returns a list of source, sourcetypes, or hosts from a specified index or distributed search peer.

- Useful for determining what is located in the indexes, based on metadata
- Does NOT present raw data
- Does respect the time picker, however snaps to the bucket times of the found event

```
|metadata type=sourcetypes | convert ctime(*Time)
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>sourceType</th>
<th>totalCount</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstTime</td>
<td></td>
<td>recentTime</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016-07-26</td>
<td>10:59:54</td>
<td>2016-07-26</td>
<td>13:07:02</td>
<td>232</td>
<td>sourceTypes</td>
</tr>
<tr>
<td>2016-07-26</td>
<td>11:03:08</td>
<td>2016-07-26</td>
<td>12:18:18</td>
<td>84</td>
<td>sourceTypes</td>
</tr>
</tbody>
</table>

union

Merges the results from two or more datasets into one dataset. One of the datasets can be a result set that is then piped into the union command and merged with a second dataset.

- Two different time ranges on same or disparate datasets
- Can be transforming or non-transforming and will do an append or multisearch depending on location of the datasets.
- Provides optimized inter-leaving of datasets based on output of the datasets

```python
union

[search earliest=@d index=main sourcetype=smartthings | eval marker="today"]
[search earliest=-1w@d latest=-1w@d+1d index=main sourcetype=smartthings | eval marker="last_week"]
| stats avg(colorTemperature) as act by device marker | where isnotnull(act)
```

<table>
<thead>
<tr>
<th>device</th>
<th>marker</th>
<th>act</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fan light</td>
<td>last_week</td>
<td>2452.492008391244</td>
</tr>
<tr>
<td>Fan light</td>
<td>today</td>
<td>2257</td>
</tr>
<tr>
<td>PlayColor</td>
<td>last_week</td>
<td>2563.875608061154</td>
</tr>
<tr>
<td>PlayColor</td>
<td>today</td>
<td>2920.941964285714</td>
</tr>
<tr>
<td>PlayColor2</td>
<td>last_week</td>
<td>2257</td>
</tr>
<tr>
<td>PlayColor2</td>
<td>today</td>
<td>2257</td>
</tr>
</tbody>
</table>
Iterative

- map, foreach

- Iterative commands do looping with events or with fields in the search pipeline.
  - EXPENSIVE commands
map

The map command is a looping operator that runs a search repeatedly for each input event or result. You can run the map command on a saved search, a current search, or a subsearch.¹

- Uses “tokens” ($field$) to pass values into the search from the previous results
- Best with either: Very small input set And/Or very specific search.
- Can take a long amount of time.
- Map is a type of subsearch
- Is “time agnostic” - time is not necessarily linear, and can be based off of the passed events, if they include time.

¹ https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Map
It takes each of the results from the rest search and searches the metadata in each index. The results are returned as a table, such as:

<table>
<thead>
<tr>
<th>sourcetype</th>
<th>index</th>
<th>values(totalCount)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveDirectory</td>
<td>msad</td>
<td>2509</td>
</tr>
<tr>
<td>GigamonForSplunk:Error</td>
<td>main</td>
<td>6</td>
</tr>
<tr>
<td>Linux:SELinuxConfig</td>
<td>os</td>
<td>2393</td>
</tr>
<tr>
<td>MSAD:NT6:Health</td>
<td>msad</td>
<td>2250</td>
</tr>
<tr>
<td>MSAD:NT6:Replication</td>
<td>msad</td>
<td>13512</td>
</tr>
<tr>
<td>MSAD:NT6:SiteInfo</td>
<td>msad</td>
<td>380</td>
</tr>
<tr>
<td>Perfmon:CPU</td>
<td>perfmon</td>
<td>7526334</td>
</tr>
</tbody>
</table>
map

This can quickly show you where there may be a sourcetype misconfigured. Why would it be in two different indexes (unless permissions play a role)? The same can be done for sources and hosts.

| rest /services/data/indexes count=0 |
| dedup title |
| fields title |
| map |
|   [ | metadata type=sourcetypes index="$title$" |
|   | eval type="$title$"] maxsearches=1000 |
| stats values(totalCount) by sourcetype type |
| rename type as index |
| stats count(index) as ci by sourcetype |
| where ci > 1 |

<table>
<thead>
<tr>
<th>sourcetype</th>
<th>ci</th>
</tr>
</thead>
<tbody>
<tr>
<td>apache_error</td>
<td>2</td>
</tr>
<tr>
<td>postfix_syslog</td>
<td>2</td>
</tr>
<tr>
<td>syslog</td>
<td>2</td>
</tr>
</tbody>
</table>
**foreach**

Runs a templated streaming subsearch for each field in a wildcarded field list.¹

- Rapidly perform evaluations and other commands on a series of fields
- Can help calculate Z scores (statistical inference comparison)
- Reduces the number of evals required

Equivalent to ...

```
| eval foo="foo" | eval bar="bar" | eval baz="baz"
```

```
... | foreach foo bar baz [eval <<FIELD>> = "<<FIELD>>"]
```

Can also use wildcards

```
| foreach foo* [ eval <<MATCHSEG1>> = "<<FIELD>>" ]
```

foobar = This, foobaz = That → bar = This, baz = That

[¹](https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Foreach)
foreach

index=_internal sourcetype=splunkd component=Metrics group=per_sourcetype_thruput
| timechart span=60m avg(kbps) as avg_kbps by series useother=f
| streamstats window=720 mean(*) as MEAN* stdev(*) as STDEV*
| foreach *
    [ eval Z_<<FIELD>> = (<<(<<FIELD>>-MEAN<<MATCHSTR>>) / STDEV<<MATCHSTR>>) ]
| fields _time Z*

<table>
<thead>
<tr>
<th>_time</th>
<th>Z_audittrail</th>
<th>Z_kvstore</th>
<th>ZSplunk_resource_usage</th>
<th>ZSplunkd</th>
<th>ZSplunkd_remote_searches</th>
<th>Z_winhostmon</th>
<th>Z_winnetmon</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-07-07 11:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016-07-07 12:00</td>
<td>0.7071</td>
<td>-0.7073</td>
<td>0.7067</td>
<td>0.7070</td>
<td>-0.7070</td>
<td>0.70710</td>
<td>-0.70711</td>
</tr>
<tr>
<td>2016-07-07 13:00</td>
<td>-0.8564</td>
<td>0.0849</td>
<td>-0.189</td>
<td>0.171</td>
<td>0.84928</td>
<td>0.05686</td>
<td>1.12954</td>
</tr>
<tr>
<td>2016-07-07 14:00</td>
<td>0.7051</td>
<td>0.7910</td>
<td>0.9665</td>
<td>1.330</td>
<td>0.8034</td>
<td>1.43314</td>
<td>0.509499</td>
</tr>
<tr>
<td>2016-07-07 15:00</td>
<td>0.8939</td>
<td>-0.9532</td>
<td>1.550</td>
<td>-0.589</td>
<td>0.8402</td>
<td>1.35145</td>
<td>-1.20408</td>
</tr>
<tr>
<td>2016-07-07 16:00</td>
<td>0.6538</td>
<td>-0.046</td>
<td>-0.0504</td>
<td>-0.630</td>
<td>-1.8309</td>
<td>-0.60302</td>
<td>-0.430569</td>
</tr>
<tr>
<td>2016-07-07 17:00</td>
<td>0.0813</td>
<td>-1.030</td>
<td>1.411</td>
<td>-1.118</td>
<td>-0.87633</td>
<td>0.779357</td>
<td>-0.13692</td>
</tr>
<tr>
<td>2016-07-07 18:00</td>
<td>-1.421</td>
<td>0.314</td>
<td>1.369</td>
<td>-0.8766</td>
<td>-0.4585</td>
<td>0.973972</td>
<td>-2.01103</td>
</tr>
<tr>
<td>2016-07-07 19:00</td>
<td>-0.0174</td>
<td>1.220</td>
<td>1.425</td>
<td>0.6707</td>
<td>0.1472</td>
<td>0.52094</td>
<td>-1.40475</td>
</tr>
<tr>
<td>2016-07-07 20:00</td>
<td>-1.499</td>
<td>-0.526</td>
<td>1.453</td>
<td>-1.356</td>
<td>0.1022</td>
<td>0.56083</td>
<td>-0.258726</td>
</tr>
</tbody>
</table>
```bash
index=_internal sourcetype=splunkd component=Metrics group=per_sourcetype_thruput
| timechart span=60m avg(kbps) as avg_kbps by series useother=f
| streamstats window=720 mean(*) as MEAN* stdev(*) as STDEV*
| foreach *
  [ eval Z_<<FIELD>> = ((<<FIELD>>-MEAN<<MATCHSTR>>) / STDEV<<MATCHSTR>>) ]
| fields _time Z*
| eval upper=3, lower=3
```
Statistics

• contingency, tstats, xyseries, streamstats, eventstats, autoregress, untable

• These are commands that build tables, evaluate sums, counts or other statistical values
**untable**

Converts results from a tabular format to a format similar to stats output. This command is the inverse of xyseries.

- Allows you to “undo” a table
  - Generate a table using a field name as row values
  - Great for doing additional evals and calculations after a transforming command

```plaintext
... | timechart avg(delay) by host | untable _time host avg_delay
```

```
index=main sourcetype=smartthings| timechart avg(battery) by device
```

<table>
<thead>
<tr>
<th>_time</th>
<th>Kitchen ceiling water sensor</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-08-10 07:00:00</td>
<td>100</td>
</tr>
<tr>
<td>2017-08-10 07:30:00</td>
<td>100</td>
</tr>
<tr>
<td>2017-08-10 08:00:00</td>
<td>100</td>
</tr>
<tr>
<td>2017-08-10 08:30:00</td>
<td>100</td>
</tr>
<tr>
<td>2017-08-10 09:00:00</td>
<td>100</td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th>_time</th>
<th>host</th>
<th>avg_delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-08-10 07:00:00</td>
<td>Kitchen ceiling water sensor</td>
<td>100</td>
</tr>
<tr>
<td>2017-08-10 07:30:00</td>
<td>Kitchen ceiling water sensor</td>
<td>100</td>
</tr>
<tr>
<td>2017-08-10 08:00:00</td>
<td>Kitchen ceiling water sensor</td>
<td>100</td>
</tr>
<tr>
<td>2017-08-10 08:30:00</td>
<td>Kitchen ceiling water sensor</td>
<td>100</td>
</tr>
<tr>
<td>2017-08-10 09:00:00</td>
<td>Kitchen ceiling water sensor</td>
<td>100</td>
</tr>
</tbody>
</table>
In statistics, **contingency tables** are used to record and analyze the relationship between two or more (usually categorical) variables. \(^1\)

A contingency table is a table showing the distribution (count) of one variable in rows and another in columns, and is used to study the association between the two variables.

Contingency is best used where there is a single value of a variable per event.

- Web Analytics - Browsers with Versions
- Demographics - Ages with Locations or Genders
- Security - Usernames with Proxy Categories
- Great to compare categorical fields

### Contingency Table

<table>
<thead>
<tr>
<th>block_type</th>
<th>STONE_PICKAXE</th>
<th>IRON_PICKAXE</th>
<th>NETHERRACK</th>
<th>SMOOTH_BRICK</th>
<th>POWERED_RAIL</th>
<th>TORCH</th>
<th>SMOOTH_STAIRS</th>
</tr>
</thead>
<tbody>
<tr>
<td>COAL_ORE</td>
<td>501</td>
<td>381</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>IRON_ORE</td>
<td>398</td>
<td>270</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GOLD_ORE</td>
<td>0</td>
<td>77</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>QUARTZ_ORE</td>
<td>10</td>
<td>0</td>
<td>17</td>
<td>6</td>
<td>5</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>GLOWING_REDSTONE_ORE</td>
<td>0</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DIAMOND_ORE</td>
<td>0</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
xyseries

Converts results into a format suitable for graphing.

Xyseries can help you build a chart with multiple data series.

- Email Flow [ xyseries email_domain email_direction count ]
- One to Many relationships [ example Weather Icons ]
- Any data that has values INDEPENDENT of the field name
- host=myhost domain=splunk.com metric=kbps metric_value=100
- xyseries domain metric metric_value
- Works great for Categorical Field comparison

```plaintext
'weather_data' | xyseries icon weather weather

<table>
<thead>
<tr>
<th>Icon</th>
<th>Clear</th>
<th>Fog</th>
<th>Haze</th>
<th>Heavy Rain</th>
<th>Heavy Thunderstorms and Rain</th>
<th>Light Rain</th>
<th>Light Thunderstorms and Rain</th>
<th>Mostly Cloudy</th>
<th>Overcast</th>
<th>Partly Cloudy</th>
<th>Rain</th>
<th>Scattered Clouds</th>
<th>Thunderstorms and Rain</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear</td>
<td>Clear</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cloudy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fog</td>
<td>Fog</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hazy</td>
<td>Haze</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mostlycloudy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Mostly Cloudy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>partycloudy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Partly Cloudy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rain</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Rain</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tstorms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Thunderstorms and Rain</td>
</tr>
</tbody>
</table>
```
eventstats

Adds summary statistics to all search results. ¹

```
eventtype=splunkcraft2016 player=alacercogitatus
| eventstats dc(block_type) as dc_block_type by player
| table player dc_block_type
```

[¹] https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Eventstats
streamstats

Adds cumulative summary statistics to all search results in a streaming manner. The streamstats command calculates statistics for each event at the time the event is seen.\(^1\)

```
eventtype=splunkcraft2016 player=alacercogitatus
| streamstats dc(block_type) as dc_block_type by player
| table player dc_block_type
```

Use the tstats command to perform statistical queries on indexed fields in tsidx files. The indexed fields can be from normal index data, tscollect data, or accelerated data models.¹

- Can only be used on indexed fields. EXTRACTED FIELDS WILL NOT WORK
- Quick way to access metadata or accelerated data (from data models or saved searches)
- Respects the time picker and default searched indexes

```
| tstats count by sourcetype index
| stats dc(index) as ci by sourcetype
```

¹ https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Tstats
```
tstats count from datamodel=Minecraft by All_Minecraft.block.item _time span=15m
| eventstats count as block_count by All_Minecraft.block.item
| search block_count > 15
| xseries _time All_Minecraft.block.item count
```
mstats

Use the mstats command to analyze metrics. This command performs statistics on the measurement, metric_name, and dimension fields in metric indexes.¹

- New in 7.0
- Can only be used on metric indexes
- Respects the time picker

```
| rest /services/catalog/metricstore/metrics
| table title
```

First, find all the metric names in the store

¹ https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Mstats
mstats

```
mstats avg(_value) as "Avg" WHERE metric_name="cpu.system.value" index=metrics span=1m
```
autoregress

Prepares your events for calculating the autoregression, or the *moving average*, by copying one or more of the previous values for *field* into each event.  

A Moving Average is a succession of averages calculated from successive events (typically of constant size and overlapping) of a series of values.

- Allows advanced statistical calculations based on previous values
- Moving Averages of numerical fields
- Network bandwidth trending - kbps, latency, duration of connections
- Web Analytics Trending - number of visits, duration of visits, average download size
- Malicious Traffic Trending - excessive connection failures

```
<table>
<thead>
<tr>
<th>name</th>
<th>kb</th>
<th>kb_p1</th>
</tr>
</thead>
<tbody>
<tr>
<td>index_thruput</td>
<td>67.896484</td>
<td>67.896484</td>
</tr>
<tr>
<td>index_thruput</td>
<td>130.467773</td>
<td>130.467773</td>
</tr>
<tr>
<td>index_thruput</td>
<td>96.457031</td>
<td>96.457031</td>
</tr>
<tr>
<td>index_thruput</td>
<td>172.701172</td>
<td>172.701172</td>
</tr>
<tr>
<td>index_thruput</td>
<td>102.154297</td>
<td>102.154297</td>
</tr>
<tr>
<td>index_thruput</td>
<td>26.770508</td>
<td>26.770508</td>
</tr>
<tr>
<td>index_thruput</td>
<td>35.422852</td>
<td>35.422852</td>
</tr>
<tr>
<td>index_thruput</td>
<td>27.778320</td>
<td>27.778320</td>
</tr>
<tr>
<td>index_thruput</td>
<td>35.808594</td>
<td>35.808594</td>
</tr>
<tr>
<td>index_thruput</td>
<td>26.131836</td>
<td>26.131836</td>
</tr>
<tr>
<td>index_thruput</td>
<td>27.364258</td>
<td>26.131836</td>
</tr>
</tbody>
</table>
```
SPL Hacks (SPLacks)

• Eval function with a stats/timechart command
  • Indirect Referencing

• These are not actual commands, but are more along the lines of a hack
Inline Timeline Eval

- You can use an eval statement in a timechart command

```plaintext
index=_internal sourcetype=splunkd source=*metrics.log group=per_sourcetype_thruput
| eval ev2 = kb / ev
| timechart span=1h eval(avg(kb) / avg(ev)) as "AVG KB per Event - 2" avg(ev2) as "AVG KB per Event - 1"
```

<table>
<thead>
<tr>
<th>AVG KB per Event - 2</th>
<th>AVG KB per Event - 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.10579</td>
<td>18.105794</td>
</tr>
<tr>
<td>18.09486</td>
<td>18.094857</td>
</tr>
<tr>
<td>18.04806</td>
<td>18.048062</td>
</tr>
<tr>
<td>17.91012</td>
<td>17.910115</td>
</tr>
<tr>
<td>17.77742</td>
<td>17.777419</td>
</tr>
</tbody>
</table>

- There is a difference in significant digits.
- Must rename the field.
Dynamic Eval (aka Indirect Reference)

- Not a search command
- NOTE: It’s a hack, so it might not work in the future.
- Works great for perfmon sourcetypes, but can be applied to any search

```
sourcetype=perfmon: dns earliest=-1h | eval cnt_{counter} = Value | stats avg(cnt_*) as *
```

The Raw Event
07/29/2016 06:07:01.973 -0800
collection=DNS
object=DNS
counter="TCP Message Memory"
instance=0
Value=39176

The New Event
07/29/2016 06:07:01.973 -0800
collection=DNS
object=DNS
counter="TCP Message Memory"
instance=0
Value=39176
cnt_TCP_Message_Memory = 39176
Dynamic Eval - Subsearch

- Not a search command
- NOTE: It’s a Splunk hack, so it might not work in the future.

```plaintext
index=_internal sourcetype=splunkd source=*metrics.log group=per_sourcetype_thruput | eval sub_host = replace(
    [] | metadata type=hosts index=_internal
    | head 1
    | rename host as query
    | fields query
    | eval query="" .query."" ,"" ,"" )
| eval subsearch = if(host==sub_host,"setting_1","setting_2")
```

<table>
<thead>
<tr>
<th>host</th>
<th>sub_host</th>
<th>subsearch</th>
</tr>
</thead>
<tbody>
<tr>
<td>1f6cc49cc777</td>
<td>1f6cc49cc777</td>
<td>setting_1</td>
</tr>
</tbody>
</table>
CLI Commands

- Commands that are run from the command line to help extract data, pull configurations, etc.
CLI Commands

• $SPLUNK_HOME/bin/splunk reload monitor
  - reloads monitor configuration, starts consuming data
  - "add/edit/enable/disable monitor" changes since last reload or Splunk restart

• Why?
  - Adding a new app/datasource
  - Don’t have to restart UF/HF for monitor configs (syslog comes to mind)
CLI Commands

• `$SPLUNK_HOME/bin/splunk cmd pcregextest`
  - Useful for testing regular expressions for extractions

```
splunk cmd pcregextest mregex="[[ip:src_]] [[ip:dst_]]" ip="(?<ip>\d+[[dotnum]]{3})" dotnum="\./d+
  test_str="1.1.1.1 2.2.2.2"
```

Original Pattern: '[[ip:src_]] [[ip:dst_]]'
Expanded Pattern: '(?<src_ip>\d+(?:\.\d+){3}) (?<dst_ip>\d+(?:\.\d+){3})'

SUCCESS - match against: '1.1.1.1 2.2.2.2'

#### Capturing group data ####

<table>
<thead>
<tr>
<th>Group</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>src_ip</td>
<td>1.1.1.1</td>
</tr>
<tr>
<td>2</td>
<td>dst_ip</td>
<td>2.2.2.2</td>
</tr>
</tbody>
</table>
CLI Commands

• `$SPLUNK_HOME/bin/splunk btool`
  
  Btool allows you to inspect configurations and what is actually being applied to your sourcetypes

• `splunk cmd btool --debug props list wunderground | grep -v "system/default"

/opt/splunk/etc/apps/TA-wunderground/default/props.conf [wunderground]

/opt/splunk/etc/apps/TA-wunderground/default/props.conf KV_MODE = json

/opt/splunk/etc/apps/TA-wunderground/default/props.conf MAX_EVENTS = 100000

/opt/splunk/etc/apps/TA-wunderground/default/props.conf MAX_TIMESTAMP_LOOKAHEAD = 30

/opt/splunk/etc/apps/TA-wunderground/default/props.conf REPORT-extjson = wunder_ext_json

/opt/splunk/etc/apps/TA-wunderground/default/props.conf SHOULD_LINEMERGE = true

/opt/splunk/etc/apps/TA-wunderground/default/props.conf TIME_PREFIX = observation_epoch

/opt/splunk/etc/apps/TA-wunderground/default/props.conf TRUNCATE = 1000000
CLI Commands

• `$SPLUNK_HOME/bin/splunk cmd python`
  • Executes the Splunk python interpreter

• `$SPLUNK_HOME/splunk cmd splunkd print-modinput-config <mi> <mi>://<stanza>`
  • Prints the modularinput configuration

• `$SPLUNK_HOME/splunk cmd splunkd print-modinput-config ga ga://kyleasmith.info | /opt/splunk/bin/splunk cmd python /opt/splunk/etc/apps/GoogleAppsForSplunk/bin/ga.py`
  • Returns and prints to the screen the modular input output (results of API calls)
Where to next?

- Splunk Search and Performance Improvements
  - Alex James, Manan Brahmkshatriya, Tuesday, Sept 26, 3:30 PM, Ballroom A

- Regex in your SPL
  - Michael Simko, Wednesday Sep 27, 11 AM, Room 150AB

- Splunk, Docs, and You
  - Rich Mahlerwein, Christopher Gales, Wednesday, Sep 27, 2:15PM, Room 103 AB

- Speed up your searches
  - Satoshi Kawasaki, Thursday, Sep 28, 10:30 AM, Room 150AB
Resources and Questions

- IRC #splunk on efnet.org (look for alacer)
- docs.splunk.com
- answers.splunk.com (I’m alacercogitatus)
- wiki.splunk.com
- Slack! Join a User Group! (https://splunk-usergroups.signup.team)
- The Splunk Trust - We are here to help! (find us by our fez!)
Thank You

Don't forget to rate this session in the .conf2017 mobile app

And JOIN the Community!