

The “Gotchas” of Splunk (Users Beware)!

Simple mistakes to make
that are easy to overlook,
difficult to diagnose, and
cause significant problems which,
in the worst-case scenario,
could even get you fired!

AKA

How I came to leave my previous employment
and start my own company!

splunk>

.conf19



Gregg Woodcock

President, Splunxter, Inc.
#3 on Splunk Answers!



Noah Woodcock

Sr. Consultant, Splunxter, Inc.
#134969 on Splunk Answers!

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. © 2019 Splunk Inc. All rights reserved.

What's a "Gotcha" anyway?

"Reports that say that something hasn't happened are always interesting to me, because as we know, there are **known knowns** - there are things we know we know. We also know there are **known unknowns** - that is to say we know there are some things we do not know. But there are also **unknown unknowns** - the ones **we don't know we don't know**. And if one looks throughout the history of our country and other free countries, it is the latter category that tend to be the difficult ones."

– United States Secretary of Defense Donald Rumsfeld in answer to a question at a U.S. Department of Defense (DoD) news briefing on February 12, 2002 about the lack of evidence linking the government of Iraq with the supply of weapons of mass destruction (WoMD) to terrorist groups.

.conf19

splunk>



Gotcha #1

The negative & insensitive world of SPL

<u>SPL</u>	<u>Result ("NOT" is <i>NOT</i> the same as "!=")</u>
NOT gregg NOT "Gregg" NOT GrEgG	Drops events containing string 'gregg' any where/case. (Keeps events <i>only</i> if '_raw' does <i>NOT</i> contain 'gregg'; some fields may have value 'gregg'.)
NOT user=gregg NOT user="gregg" NOT user=gReGg	Drops events where field 'user' exists AND has value 'gregg' <u>BUT ALSO keeps events where field 'user' does not exist!</u> (Drops events <i>even if</i> '_raw' contains 'gregg'.)
user != gregg user != Gregg	Drops events where field 'user' exists & has value 'gregg' <u>BUT ALSO drops events where field 'user' does not exist!</u> (Drops events <i>even if</i> '_raw' contains 'gregg'.)
NOT "user=gregg" NOT "User=GrEgg"	Drops events containing string 'user=gregg' any where/case. (Keeps events where field 'user' has value 'gregg', unless '_raw' also contains 'user=gregg'.)

Gotcha #2

Where is the Sensitivity?

THE SETUP: You are trying to count very particular events:

```
sourcetype=MyEvents MyField= "MyCaseSensitiveValue" | stats count
```

THE PROBLEM: You are receiving too many results and your investigation reveals it is because your base search is polluted by false-positive (*case-insensitive*) matches.

Gotcha #2

Where is the Sensitivity?

THE SOLUTION: Use CASE() or any other post-pipe comparison/filter:

```
sourcetype=MyEvents MyField=CASE(MyCaseSensitiveValue)
```

```
sourcetype=MyEvents MyField="MyCaseSensitiveValue"  
| where MyField="MyCaseSensitiveValue"
```

```
sourcetype=MyEvents MyField="MyCaseSensitiveValue"  
| where match(MyField, "MyCaseSensitiveValue")
```

```
sourcetype=MyEvents MyField="MyCaseSensitiveValue"  
| regex MyField="MyCaseSensitiveValue"
```

```
sourcetype=MyEvents MyField="MyCaseSensitiveValue"  
| eval caseInsensitiveMatch=if(MyField="MyCaseSensitiveValue",1,0)  
| search caseInsensitiveMatch=1
```

THE EXPLANATION: Splunk is case-sensitive for string-literal values (not field names) everywhere except in the 'search' command (base search).

NOTE: The first method (using "CASE()") is vastly superior to the others.

Gotcha #3

SPL can't escape from wildcards

THE SETUP: You are trying to filter out header/comment lines that are full of asterisks ("*****"):

*sourcetype=MyEvents NOT ******

*sourcetype=MyEvents NOT "*****"*

*sourcetype=MyEvents NOT '*****'*

sourcetype=MyEvents NOT ||*|*|*|*|*|*|*|**

sourcetype=MyEvents NOT "||*|*|*|*|*|*|*|*"*

sourcetype=MyEvents NOT '||*|*|*|*|*|*|*|*'*

THE PROBLEM: For some reason, no matter what you do, everything stays or everything disappears!

Gotcha #3

SPL can't escape from wildcards

THE SOLUTION: You must use "match" (RegEx), "like" (SQL), or similar streamable command function instead:

```
sourcetype=MyEvents  
| where NOT match(_raw, "^|*|*|*|*|*|*|*|*|*|*")
```

```
sourcetype=MyEvents  
| where NOT like(_raw, "*****%")
```

```
sourcetype=MyEvents  
| regex _raw!="^|*|*|*|*|*|*|*|*|*|*"
```

THE EXPLANATION: I don't know the reasoning behind it but asterisks ("*") cannot be escaped in the general base search (they will always be treated as wildcard characters).

Gotcha #4

Wildcard-ignorant & literal "eval"

THE SETUP: You have been asked to find out how many times something happened in your data:

```
sourcetype=MyEvents  
| stats count count(eval(MyField="123*")) AS MyCount  
| eval MyPct = 100 * MyCount / count
```

THE PROBLEM: For some reason, the number is way too low. Worse yet, when you run the math directly (without eval), you get a much larger answer that seems correct:

```
sourcetype=MyEvents MyField="123"  
| stats count
```

Gotcha #4

Wildcard-ignorant & literal "eval"

THE SOLUTION: With "eval" you may NEVER use wildcards; you must use "match" (RegEx), or "like" (SQL) instead:

```
sourcetype=MyEvents  
| stats count count(eval(match(MyField, "^123*"))) AS MyCount  
| eval MyPct = 100 * MyCount / count
```

```
sourcetype=MyEvents  
| stats count count(eval(like(MyField, "123%"))) AS MyCount  
| eval MyPct = 100 * MyCount / count
```

THE EXPLANATION: Splunk "eval" and "where" always treat asterisks ("*") as string literals (which is why you do not get an error, just unexpectedly incorrect results), never as wildcards. This is the converse of the previous gotcha!

Gotcha #5

It's the same 'case' even 'if' you can't see 'eval'!

THE SETUP: You have been asked to find out how many times something happened in your data:

```
sourcetype=MyEvents  
| eval type=case(MyField="123*", "123", true(), "Other")  
| stats count AS MyCount BY type
```

THE PROBLEM: For some reason, everything buckets to "Other". Strangely, when you run the math directly (without eval), you get a non-zero answer that seems correct:

```
sourcetype=MyEvents MyField="123*" | stats count
```

Gotcha #5

It's the same 'case' even 'if' you can't see 'eval'!

THE SOLUTION: With "case" (and "if") you may NEVER use wildcards; use "match" (RegEx), or "like" (SQL) instead:

```
sourcetype=MyEvents  
| eval type=case(match(MyField, "^123*"), "123", true(), "Other")  
| stats count AS MyCount BY type
```

```
sourcetype=MyEvents  
| eval type=case(like(MyField, "123%"), "123", true(), "Other")  
| stats count AS MyCount BY type
```

THE EXPLANATION: The splunk "case" and "if" commands are just like the "eval" command and in fact are driven by the same "eval" and "where" code. This is exactly the same as the previous gotcha!

Gotcha #6

Time, time, what is time?

THE SETUP: A search has been saved and scheduled to run every day, in the middle of the night, over a time range that covers the previous day.

```
sourcetype=MyEvents earliest=-1d@d latest=@d
```

THE PROBLEM: Some details of this report are being compared against a report that is generated from a non-splunk system-of-record that is using the same event data (or legitimate analog) and same time range, but the numbers, although always very close, never exactly match.

Gotcha #6

Time, time, what is time?

THE SOLUTION: Have the user that owns the saved search change his "Time zone" setting.

THE EXPLANATION: Every scheduled job runs AS A USER (the user that saved the search and owns it). Each user has a "Time zone" setting inside his profile (under My User Name -> Edit account -> Time zone). If, for example, you tell splunk to run a search at "3AM", you are actually saying run it at "3AM as defined by this user's Time zone setting" and, much more importantly, over the "day" (-1d@d -> 0d@d) also as defined by that user's "Time zone" setting. The question is: what (or rather, "when") is "a day" (as regards the snap-to part)? If the user uses "CST", then the window that defines "all of yesterday" is several hours different than the window that would be defined if he had used "UTC". This is an even bigger problem if the saved search is a populating search for a Summary Index!

Gotcha #7

Ghost in the machine!

THE SETUP: You have many reports that are being generated from saved searches and these have always been just fine.

THE PROBLEM: After a major restructuring that included both layoffs and personnel realignments, some reports are returning bad results. There are even rumors of sabotage by disgruntled employees that were adversely effected! You find this hard to believe but why are so many reports doing such strange things?

Gotcha #7

Ghost in the machine!

THE SOLUTION: Repair any saved searches that are scheduled but no longer have proper ownership/permission (each owner must have the appropriate roles: those that the user previously had). Use this search to find them:

```
| rest /servicesNS/-/-/saved/searches  
| where is_scheduled=1 AND disabled=0 AND  
  (isnull(next_scheduled_time) OR ($eai:acl.owner$="nobody"))  
| fields eai:acl.owner cron_schedule is_scheduled eai:acl.app  
  next_scheduled_time title updated splunk_server disabled
```

THE EXPLANATION: Scheduled searches run in context of an owner who must exist and have appropriate roles. If a search is "active" and "scheduled" but has no "next_scheduled_time" or "nobody" owns it, then you have trouble. If it is owned by "nobody" then the user that originally owned it has been deleted and the permissions he had (determined by his "roles", frequently matched to Active Directory roles) are no longer in place so the search may behave differently now than it did previously when run as the proper owner.

Gotcha #8

What you don't see *can* hurt you!

THE SETUP: More and more, you are getting reports of "bad searches", or queries that "don't match results from a separate run". In every case, when you re-type the search string manually, it works. Visually the strings are identical, but apparently there are invisible characters introduced at some point in the crafting of the string (most likely through cut-and-paste) the break it.

THE PROBLEM: When the search fails completely, the problem is apparent and it gets discovered and fixed. Sometimes, however, it's a "silent failure" and returns wrong results that are not easily noticed until "too late". For example, if the search has a field filter in like "myField!=someValue" but your search does not return events that you know exist, inspect the search job. The Job Inspector may reveal that Splunk used "myField!=someValue\u00a0" instead (where the character "\u00a0" is unprintable/invisible everywhere except in the text shown by the Job Inspector, which shows a hexadecimal encoding of it).

KEY TAKEAWAY: Use the job inspector when things are strange; it is an incredibly powerful investigative tool helps you learn how Splunk works "under the sheets".

Gotcha #8

What you don't see *can* hurt you!

THE SOLUTION: Find the broken searches and fix each one by re-typing it manually:

```
| rest /servicesNS/-/-/saved/searches/
| rex field=search "(?<NonStandardCharacter>[^\s~\r\n]+)"
| where isnotnull(NonStandardCharacter)
| rex field=search "(?<AroundNonStandardCharacter>.{4}[^\s~\r\n]+.{4})"
| search NOT title="DMC Asset - Build Full"
| fields title eai:acl.owner search
| rename title AS Name eai:acl.app AS SplunkAppLocation
search AS SearchQuery
```

THE EXPLANATION: This search finds search strings with unprintable characters in the SPL. This should probably be scheduled as an alert and also run against recently run searches (not just saved searches) with the appropriate admin response, respectively: either saved search repair or user education.

Gotcha #9

Where is (literally) not Search

THE SETUP: You are trying to count very particular events:

```
sourcetype=MyEvents MyField="*" "Other Field" ="*"  
| where MyField="Other Field"  
| stats count
```

THE PROBLEM: You are receiving counts that are WAY off and your investigation reveals it is because your results are missing most of your data that you know is there.

Gotcha #9

Where is (literally) not Search

THE SOLUTION: Enclose your multi-word field name inside *dollar-signs* or *single-quotes* (I prefer the former):

```
sourcetype=MyEvents MyField="*" "Other Field" =""  
| where MyField=$Other Field$ OR MyField='Other Field'  
| stats count
```

THE EXPLANATION: The main difference between 'where' and 'search' (besides the nature of case-in/sensitivity previously discussed), is that 'where' presumes the right-hand-value (RHV) of the equals-sign to be another field name but it can be convinced otherwise if you enclose the RHV in double-quotes. The problem here is that using enclosing double-quotes is also the most common way to specify a multi-word (whitespace polluted) field name, too! So which interpretation wins out in this ambiguous case? In such a situation, the RHV will always be treated as a string literal. To force it to be treated as a field name instead, use the less-common (but also less-ambiguous) method of specifying a field name of enclosing in bounding dollar-signs or single-quotes.

Gotcha #10

Don't add my cat, Dog!

THE SETUP: You need to create a new field by concatenating 2 other fields:

```
sourcetype=MyEvents  
| eval CatField = MyField + OtherField  
| stats count by CatField
```

THE PROBLEM: The report tested out just fine but some people are complaining that they are seeing values for CatField that are impossible (for which constituent field values for MyField and OtherField are impossible. Sure enough, you see direct evidence of the same thing when you search for it directly, with a search like this:

```
sourcetype=MyEvents  
| eval CatField = MyField + OtherField  
| stats values(MyField) values(OtherField) count by CatField
```

Gotcha #10

Don't add my cat, Dog!

THE SOLUTION: Enforce concatenation with '.' or "toString()":

```
sourcetype=MyEvents  
| eval CatField = MyField _ OtherField  
| stats count by CatField
```

```
sourcetype=MyEvents  
| eval CatField = toString(MyField) + toString(OtherField)  
| stats count by CatField
```

THE EXPLANATION: The '+' operator has 2 functions: addition *and* concatenation! So which interpretation wins out in this ambiguous case? In such a situation, it will prefer to treat the arguments as numbers to be added and will only concatenate if one (or both) is NaN (Not A Number). To force concatenation, either use the less-common (but also less-ambiguous) concatenation operator '.' or typecast the arguments with "toString()".

This content is an abbreviated version of a 3-day course covering 70+ “gotchas” including an interactive Splunk app demonstrating each one.

Email Gotchas@Splunxter.com for more details.

Q&A



Gregg Woodcock | President, Splunxter, Inc.

Noah Woodcock | Sr Consultant, Splunxter, Inc.





Thank You!

Go to the .conf19 mobile app to

**RATE THIS
SESSION**