The |format Solves the Riddle

Deep Dive on Using the Format Command to Instrument Indicators of Compromise

Trevor Ford

October 2018 | Version 0.8

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> .conf18

Who am I?

- I'm Trevor Ford
- I've been getting my Splunk on for nearly ten years now
- That makes me feel old
- I'm a cyber security analyst at the Federal Reserve Bank of San Francisco
- *The views expressed in the presentation do not necessarily present the view of the Federal Reserve Bank*

What Are We Talking About?

- This presentation is about the |format command!
- We'll go through some background explaining how the command works and how it's already used within Splunk
- We'll finish with a few interesting use cases to demonstrate what we learned
- The expected audience for this talk is advanced Splunk users
  - If you don't think you're an advanced user, you're more than welcome to stay! I hope this is still useful

Why Am I So Excited About This?

- Threat intelligence is *everywhere*
  - Commercial
  - Industry-confidential
  - Open source
  - Some guy you work with named Ronald McDonald, who is real smart and also a real clown

- Making the most of that intel requires translating a “feed” into actual Splunk searches!
  - We’re balancing effectiveness, Splunk resource consumption, and ease of maintenance
  - There’s no perfect solution – patterns are distinct from atomic indicators which are distinct from multiple-indicator compositions

|format – What's the Deal?

|format – What's the Deal?

The Documentation

- Format is a command in Splunk! (base Splunk Enterprise, not ES)
- It even has documentation!
  - This command is used implicitly by subsearches. This command takes the results of a subsearch, formats the results into a single result and places that result into a new field called search.
  - http://docs.splunk.com/Documentation/Splunk/7.1.2/SearchReference/Format

splunk> .conf18

|format – What's the Deal?

Fun Format Facts

- At its core, |format takes a table and makes it a string
- That string is stored in the field "search"
- Format is used by default every time you feed a subsearch into a search
- It has default values
- We don't have to accept that reality

|format – What's the Deal?

Here's What We're Doing

- Let's turn this:

A	B	C
D	E	F

- Into this:

A	B	C

D	E	F

splunk> .conf18

|format – What's the Deal?

Let's Break This Down

Command Syntax

```
| format  [
"<row prefix>"
"<column prefix>"
"<column separator>"
"<column end>"
"<row separator>"
"<row end>"]
```

Default Values

```
| format  [
“(“
“(“
“AND”
“)”
“OR”
“)”]
```

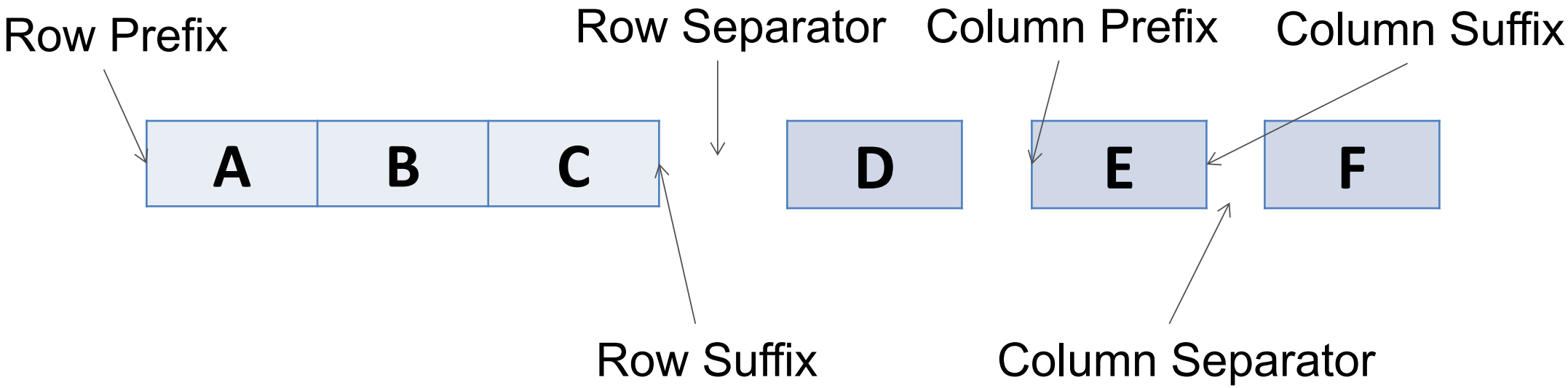
Boring Example Data

A	B	C
D	E	F

splunk> .conf18

|format – What's the Deal?

How Does It Fit Together



|format – What's the Deal?

Pull It Together!

Command Syntax

```
| format  [
"<row prefix>"
"<column prefix>"
"<column separator>"
"<column end>"
"<row separator>"
"<row end>"]
```

Default Values

```
| format  [
“(“
“(“
“AND”
“)”
“OR”
“)”]
```

Boring Example Data

A	B	C
D	E	F

((A) AND (B) AND (C)) OR ((D) AND (E) AND (F))

|format – What's the Deal?

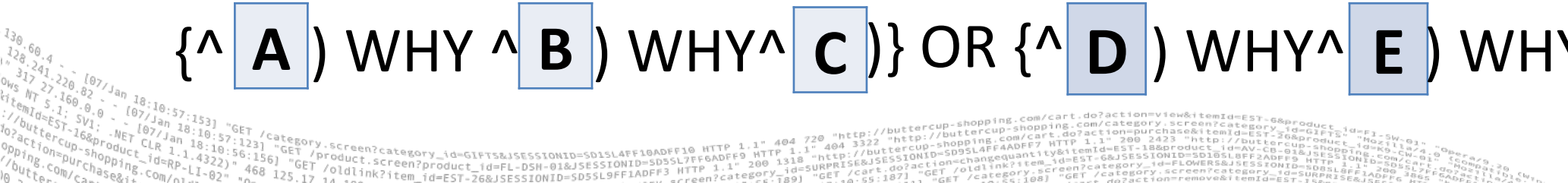
We're Too Special For Defaults

Command Syntax	New Exciting Values	Boring Example Data
| format [	| format [	
"<row prefix>"	"{"	
"<column prefix>"	"^"	
"<column separator>"	"WHY"	
"<column end>"	")"	
"<row separator>"	"OR"	
"<row end>"]	"}"]	

A	B	C
D	E	F

Note: this is gibberish.

{^ **A**) WHY ^ **B**) WHY^ **C**)} OR {^ **D**) WHY^ **E**) WHY^ **F**)}



splunk> .conf18

|format – What's the Deal?

Recap!

- What do we know about the format command?
  - It has *great* documentation
  - It turns a matrix into a single string
  - That string is in a field called "search"
  - The defaults are great, but we have the unspeakable power to ditch them

The End of a Subsearch

The End of a Subsearch

Let's Talk Defaults

- Format is automatically called with its default arguments at the end of a subsearch!
- Default arguments: separate columns with AND, separate rows with OR, use parentheses to turn each row into a tuple
- In practical terms, it turns each row into a set of criteria and feeds back a query to match any whole row
  - ((dest=8.8.8.8) AND (dest_host=google.com)) OR (dest=9.9.9.9 AND dest_host=fake.com))
- The default includes the field names/column headers in the result string

The End of a Subsearch

Who Hates Field Names? Query Hates Field Names.

- Suppose you're a rebel and you want your subsearch to feed back values but not field names
- You can use two special fields to get that done: "search" and "query"
  - A field renamed "search" or "query" will be included by |format without a field name – just the raw data
  - | rename dest_host AS query | format
    - ((dest=8.8.8.8) AND (google.com)) OR (dest=9.9.9.9 AND fake.com))

Modifying the Search String

Modifying the Search String

This is sed.

- The |format command shoves its results into the “search” field
- By default, it just passes that back
- You’re a hotshot, though, and you don’t need no stinkin defaults
- The “search” field is a string. How do we edit strings?
- That’s right, it’s sed

- |rex mode=sed field=search “s/z/ijusthatethatletter/g”

Use Cases for “Fun” and “Profit”

Use Cases

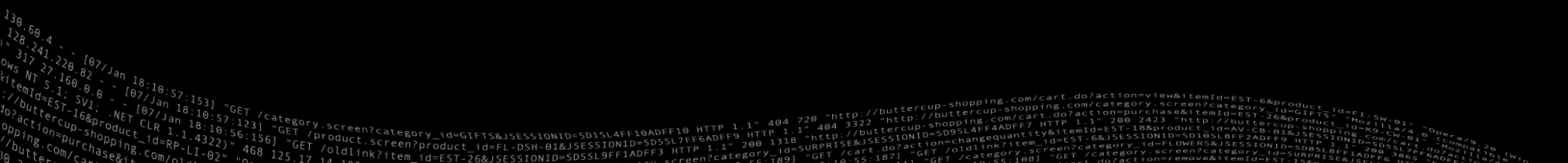
Introduction/Why Are We Here

- This presentation is about matching indicators of compromise (or indicators of bad-dude-ness, or whatever vocab term you'd like)
- All that background on |format laid out special tools for very specific situations
- Special tool situations are rare! You mostly want to do the boring stuff
- We're going to talk about lookup, about subsearching |inputlookup, and getting crazy with eval functions

Use Cases

Let's Talk About Lookup

- The base lookup command is my go-to, and it should be yours
- It's simple and straightforward
- It does, however, have some weaknesses:
  - You have to pull data, *then* match, *then* filter to matched results. This can be sloooow
  - Pattern matching is limited to wildcard characters



Use Cases

Fast IP Matching

- Quick aside: the Splunk term index!
- Splunk builds fantastic search term indexes to speed up searches
- IPv4 addresses were designed specifically to screw up term indexes:
  - The . character is a major separator. IPs have three of them
  - 1.2.3.4 becomes (1), (1.2), (1.2.3), and (1.2.3.4)
- Despite this, we can tell Splunk to jump straight to the point by searching TERM(1.2.3.4)

Use Cases

Fast IP Matching

- Situation: I have a lookup of IP addresses. The list is too long to search manually. Using |lookup is taking too long
- Desire: TERM() wrap each IP to speed things up!
- Method: |inputlookup and |format in a subsearch
- Sample SPL:
  - index=very_cool_firewall_data [
    | inputlookup baddest_dudes.csv
    | fields ipv4_address
    | rename ipv4_address AS query
    | format “(“ “TERM(“ “OR” “)” “OR” “)”]

Use Cases

Fast IP Matching

- So what happens in that search?

[| inputlookup baddest_dudes.csv	Pull in our lookup table
| fields ipv4_address	Cut it down to just the values to match
| rename ipv4_address AS query	Ditch the field name with query
| format “(“ “TERM(“ “OR” “)” “OR” “)”]	Wrap each value in TERM(<value>)

- We’re left with a list of OR-delimited search terms in the initial part of the main query – Splunk distributes the search and we’re off to the races

Use Cases

Fast IP Matching – But With Context

- That TERM() query only retrieves matching events – it doesn't include any of the context we might have documented in our lookup table of the baddest dudes
- We can still use |lookup! Even better, we're only using |lookup on events that we know will match a value in the table
- Sample SPL:
  - index=very_cool_firewall_data [
    | inputlookup baddest_dudes.csv
    | fields ipv4_address
    | rename ipv4_address AS query
    | format "(" "TERM(" "OR" ")" "OR" ")"]
    | lookup baddest_dudes.csv ipv4_address AS dest OUTPUT name_of_baddest_dude

Use Cases

Expressions so Fun They Call Them Regular

- Situation: I find myself with a growing list of regular expressions that match malicious activity (GET query string formats is a common one)
- Desire: maintain a lookup table of regexes and feed those into a search
- Method: |inputlookup combined with |format to populate an eval/where function
- Sample SPL:
  - index=very_cool_proxy_data
    | where [
    | inputlookup spooky uri_stems.csv
    | fields ioc_regex
    | format “” “match(_raw,” “OR” “)” “OR” “”
    | rex mode=sed field=search “s/ioc_regex\=//g”]

Use Cases

Throw a Little Context in There. For Flavor.

- That's a basic match, but indicators are less useful if you're trying to figure out which one did the match
- How do we combine matching logic plus context? Case statements and eval!
- Sample SPL:
  - index=very_cool_proxy_data
    | eval [
    | inputlookup spooky uri_stems.csv
    | fields ioc_id, ioc_regex
    | format "ioc_match=case(" "match(_raw," "),"" "","" ")"
    | rex mode=sed field=search "s/ioc_id\=//g"
    | rex mode=sed field=search "s/ioc_regex\=//g"]

Use Cases

Wait, What?

| eval [	Building an eval
| inputlookup spooky uri_stems.csv	Pull in our lookup of regexes
| fields ioc_id, ioc_regex	Include a table ID and the regex
| format “ioc_match=case(“ “match(_raw,” “),” “” “,” “)”	Populate the ioc_match field with the result of a case() statement that passes each lookup value into the match() function
| rex mode=sed field=search “s/ioc_id\=//g”	Clear the ID field name out of the case()
| rex mode=sed field=search “s/ioc_regex\=//g”	Clear the regex field name out of the match() functions

Use Cases

Additional Notes on That Query

- You may find yourself with an excess of escaped backslashes (\ )
- You may find yourself with improperly escaped quotations
- And you may ask yourself, how do I work here?

- sed. sed it all away

- Expect some pre/post processing if you're collecting regexes from the internet
- Even formats like STIX seem to allow enough variation to make life interesting

Use Cases

Why? Why do this thing?

- What do we really want?
  - Lookup-style management (CSV, KVS, etc)
  - Complex pattern matching
- Can we get it without these crazy queries?
  - *Kinda?*
  - Could maintain macros
  - Could maintain searches for each regex, or set of regexes
  - Could maintain field extractions!
- In my experience, lookups are the most automatable way to manage threat intelligence inside of Splunk. Your experience may differ

Last Words

Before We Go

- Operationalizing threat intelligence is hard!
  - It's worth the time to think through which data you're matching to which indicators
  - The easy stuff isn't necessarily that easy
- My advice: bust out the crazy stuff when it's needed but don't be crazy for crazy's sake

- Subject matter experts make everything easier
  - Incident responders are your best friends
  - Threat intelligence experts are also your best friends
  - Make lots of best friends

Thank You!

Don't forget to rate this session in the .conf18 mobile app