splunk> .conf2017

Regex in Your SPL

An Easy Introduction

Michael Simko | Sr. Engineer, Instructor

September 2017 | Washington, DC

Forward-Looking Statements

splunk> .conf2017

Basics of Regular Expressions

What is this Regex thing all about?

splunk> .conf2017

Regex in Splunk SPL

What's in it for me?

1. Filtering. Eliminate unwanted data in your searches
2. Matching. Advanced pattern matching to find the results you need
3. Field Extraction on-the-fly

What Is Regex?

What People Say

"A regular expression is an object that describes a pattern of characters. Regular expressions are used to ***perform pattern-matching and 'search-and-replace' functions on text.***"

– *w3schools.com*

"Regular expressions are an extremely powerful tool for manipulating text and data…
If you don't use regular expressions yet, you will..."

– *Mastering Regular Expressions, O'Rielly, Jeffery E.F. Friedl*

"A regular expression is a special text string for describing a search pattern. You can think of regular expressions as ***wildcards on steroids***."

– *Regexbuddy.com (and others – Original source unknown)*

splunk> .conf2017

Regex Basics

The Main Elements

Control Characters:	Character Types:	Operators:
^ Start of a Line	\s White Space	* Zero or More
$ End of a Line	\S Not white space	+ One or More
	\d Digit	? Zero or One
	\D Not Digit	
	\w Word Character (letter, #, or _)	
	\W Not a Word Character	

These elements work together to specify a pattern

Regex Basics

The Main Elements

Control Characters:

^ Start of a Line

$ End of a Line

Character Types:

\s White Space

\S Not white space

\d Digit

\D Not Digit

\w Word Character

\W Not Word Characters

Operators:

* Zero or More

+ One or More

? Zero or One

Sample Regex: ^\d+\s\w+\d+\s\d+:\d+:\d+

: is the literal character colon

\s without a + or * is a single space

\w+ is one or more word characters

\d+ is one or more digits

^ Regex is Anchored to the beginning of the line

splunk> .conf2017

Regex Basics

The Main Elements

Control Characters:
^ Start of a Line
$ End of a Line

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Operators:
* Zero or More
+ One or More
? Zero or One

Sample Regex: ^\d+\s\w+\d+\s\d+:\d+:\d+

Matching String: 22 Aug 2017 18:45:20 On this date, Michael made BBQ references

splunk> .conf2017

Regex Basics

To Protect and Give Options

Control Characters:
^ Start of a Line
$ End of a Line

Special Characters:
| Alternative / “or”

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Protection Characters:
\ The next character is a literal

Special Characters:
To give multiple options: |
The pipe character
(also called “or”)

Protecting Characters:
To escape or protect special characters: \
The Backlash or back-whack

Protect periods, [],(),{}, etc when you want to use the literal character

Regex Basics

To Protect and Give Options

Control Characters:
^ Start of a Line
$ End of a Line

Special Characters:
| Alternative / "or"

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Protection Characters:
\ The next character is a literal

Regex: **Indiana|Purdue**

Purdue 8w 3l .727 19w 5l .792
Indiana 5w 4l .500 15w 8l .652

Regex: **\d+\.\d+\.\d+\.\d+**

Login Failure From **192.168.12.145**
Login Success From **10.35.36.37**

(we'll do the above a different way later)

Regex Basics

Only Some May Pass

Control Characters:
^ Start of a Line
$ End of a Line

Special Characters:
| Alternative / “or”

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Protection Characters:
\ The next character is a literal

Inclusion Characters:
[] Include
[^] Exclude

Include Characters:
[…]

Example usage: [a-zA-Z0-9]

Exclude Characters:
[^…]

Example usage: [^]

Regex Basics

Only Some May Pass

Control Characters:
^ Start of a Line
$ End of a Line

Special Characters:
| Alternative / "or"

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Protection Characters:
\ The next character is a literal

Inclusion Characters:
[] Include
[^] Exclude

Regex: **server:[a-z0-9]+**

Regex: **server:[^]**

Keep going so long as you hit characters that are lowercase a-Z or 0-9

server:253fsf2,host=23423
server: 253fsf2,host=23423
server:253f sf2,host=23423

Go until you hit a space

Regex Basics

Say What Again

Control Characters:
^ Start of a Line
$ End of a Line

Special Characters:
| Alternative / “or”

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Protection Characters:
\ The next character is a literal

Inclusion Characters:
[] Include
[^] Exclude

Repetition:
{#} Number of Repetitions
{#,#} Range of Repetitions

Repetition is used to define the exact number of characters
Or an upper and lower boundary of acceptable characters
(or the exact number of repetitions of a pattern)

Regex Basics

Say What Again

Control Characters:
^ Start of a Line
$ End of a Line

Special Characters:
| Alternative / “or”

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Protection Characters:
\ The next character is a literal

Inclusion Characters:
[] Include
[^] Exclude

Repetition:
{#} Number of Repetitions
{#,#} Range of Repetitions

Regex: **IP: \d{3}\.\d{3}\.\d{3}\.\d{3}**

IP: 172.106.190.100
IP: 10.24.255.2
IP: 224.252.2.52

Only 1 line matched because IP format allows 1-3 digits per octet

Regex: **IP: \d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}**

IP: 172.16.19.1
IP: 10.24.255.2
IP: 224.252.2.52

All 3 lines matched since we account for the IP Address format

Regex Basics

To Protect and Give Options

Control Characters:
^ Start of a Line
$ End of a Line

Special Characters:
| Alternative / “or”

Logical Groupings:
() Wrap sets of the Regex

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Protection Characters:
\ The next character is a literal

Inclusion Characters:
[] Include
[^] Exclude

Repetition:
{#} Number of Repetitions
{#,#} Range of Repetitions

Later we’ll use these as “capture groups”

Use to specify repetition for adjacent elements in order to form patterns

Regex Basics

To Protect and Give Options

Control Characters:
^ Start of a Line
$ End of a Line

Special Characters:
| Alternative / “or”

Logical Groupings:
() Wrap sets of the Regex

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Protection Characters:
\ The next character is a literal

Inclusion Characters:
[] Include
[^] Exclude

Repetition:
{#} Number of Repetitions
{#,#} Range of Repetitions

Revisiting the IP Matching from a couple of slides ago

Alternate Regex: **IP: (\d{1,3}\.){3}\d{1,3}**

IP: 172.16.19.1
IP: 10.24.255.2
IP: 224.252.2.52

Repeats **\d{1,3}\.** three times
Then tacks on the last **\d{1,3}**

Regex Basics

The Last (Not so Basic) Element

Control Characters:
^ Start of a Line
$ End of a Line

Special Characters:
| Alternative / “or”

Logical Groupings:
() Wrap sets of the Regex

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Protection Characters:
\ The next character is a literal

Inclusion Characters:
[] Include
[^] Exclude

Repetition:
{#} Number of Repetitions
{#,#} Range of Repetitions

Named Capture Groups:
(?<CaptureGroupName>stuff)

This names the capture group (e.g., logical grouping).
Now when you return the capture, it has a name and not just “Capture Group 1”

Regex Basics

The Last (Not so Basic) Element

Control Characters:
^ Start of a Line
$ End of a Line

Special Characters:
| Alternative / “or”

Logical Groupings:
() Wrap sets of the Regex

Character Types:
\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Protection Characters:
\ The next character is a literal

Inclusion Characters:
[] Include
[^] Exclude

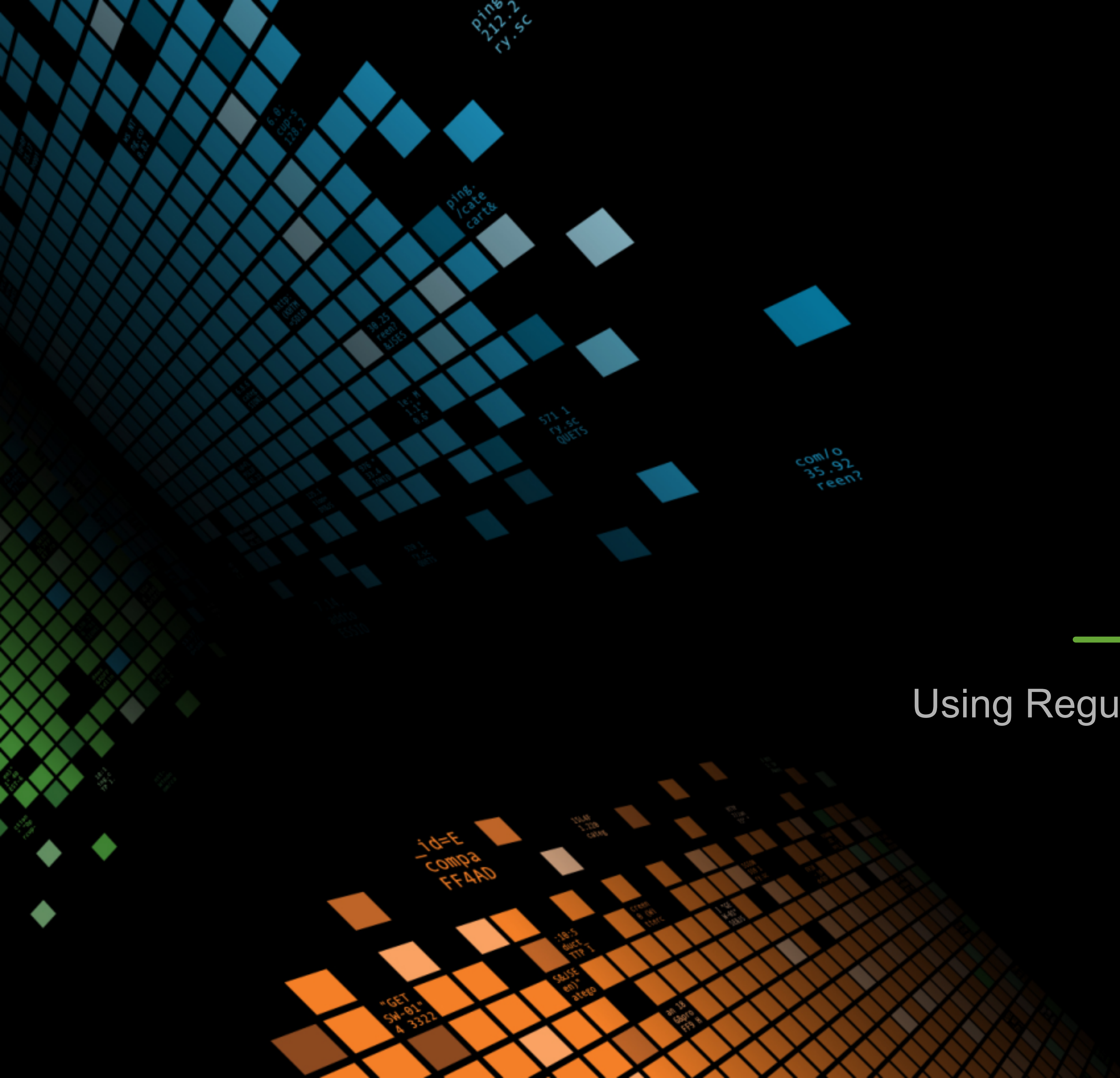
Repetition:
{#} Number of Repetitions
{#,#} Range of Repetitions

Named Capture Groups:
(?<CaptureGroupName>stuff)

Regex: **user:\s**(?**<username>[^@]+**)

Go until we hit an @
Capture as field username
Anchor off user:\s

Log 1: blah blah **user: msimko@**splunk.com
Log 2: more blah **user: michael@**kinneygroup.com



Regex in SPL

Using Regular Expressions to improve your SPL

Regex in Your SPL

Search Time Regex

- Field Extractions
  - **erex**
  - **rex**
  - Interactive Field Extractor
  - Props – Extract
  - Transforms - Report

- Evaluation
  - **Regex**
  - **match**
  - **replace**

Fields are fundamental to Splunk Search

Regex provides granularity when evaluating data

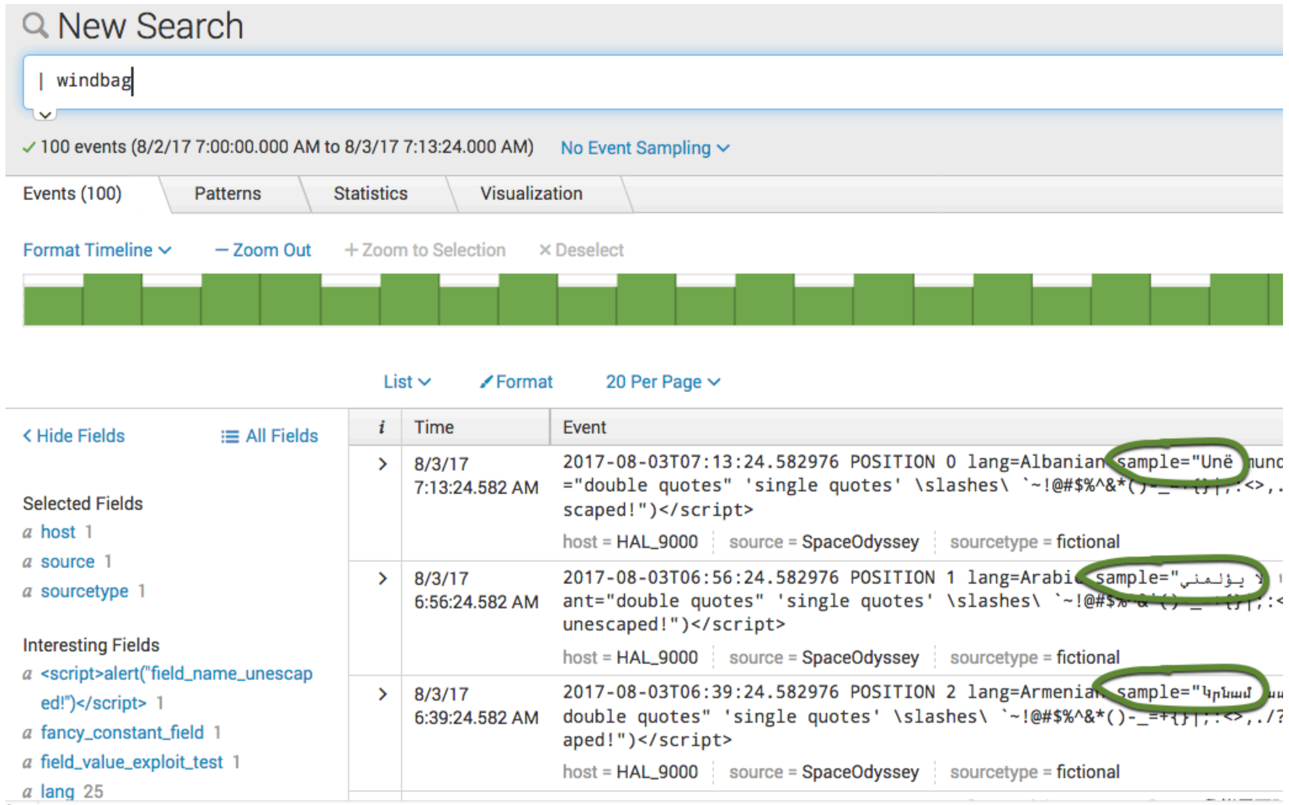
Field Extractions

On the fly (No need to work ahead)

erex Command

Field Extractions Using Examples

Use Splunk to generate regular expressions by providing a list of values from the data.



New Search

```
| windbag
```

✓ 100 events (8/2/17 7:00:00.000 AM to 8/3/17 7:13:24.000 AM) No Event Sampling

Events (100) | Patterns | Statistics | Visualization

Selected Fields: host 1, source 1, sourcetype 1

Interesting Fields: <script>alert("field_name_unescaped!")</script> 1, fancy_constant_field 1, field_value_exploit_test 1, lang 25

i	Time	Event
>	8/3/17 7:13:24.582 AM	2017-08-03T07:13:24.582976 POSITION 0 lang=Albanian sample="Unë munc ="double quotes" 'single quotes' \slashes\ `~!@#$%^&*()-_=+{}|;:<>,. scaped!")</script> host = HAL_9000 source = SpaceOdyssey sourcetype = fictional
>	8/3/17 6:56:24.582 AM	2017-08-03T06:56:24.582976 POSITION 1 lang=Arabic sample="لا يؤلمني ant="double quotes" 'single quotes' \slashes\ `~!@#$%^&*()-_=+{}|;:< unescaped!")</script> host = HAL_9000 source = SpaceOdyssey sourcetype = fictional
>	8/3/17 6:39:24.582 AM	2017-08-03T06:39:24.582976 POSITION 2 lang=Armenian sample="Կրնամ ա double quotes" 'single quotes' \slashes\ `~!@#$%^&*()-_=+{}|;:<>,./? aped!")</script> host = HAL_9000 source = SpaceOdyssey sourcetype = fictional

- Scenario: Extract the first word of each sample phrase from | windbag
  - Step 1, find the samples
  - Step 2, extract the field

splunk> .conf2017

erex Command

Field Extractions Using Examples

Erex Command: …| erex <newFieldName> examples="example1,example2"

| **windbag** | erex **firstwords** examples="**Unë, يؤلمن, Կրնամ**"

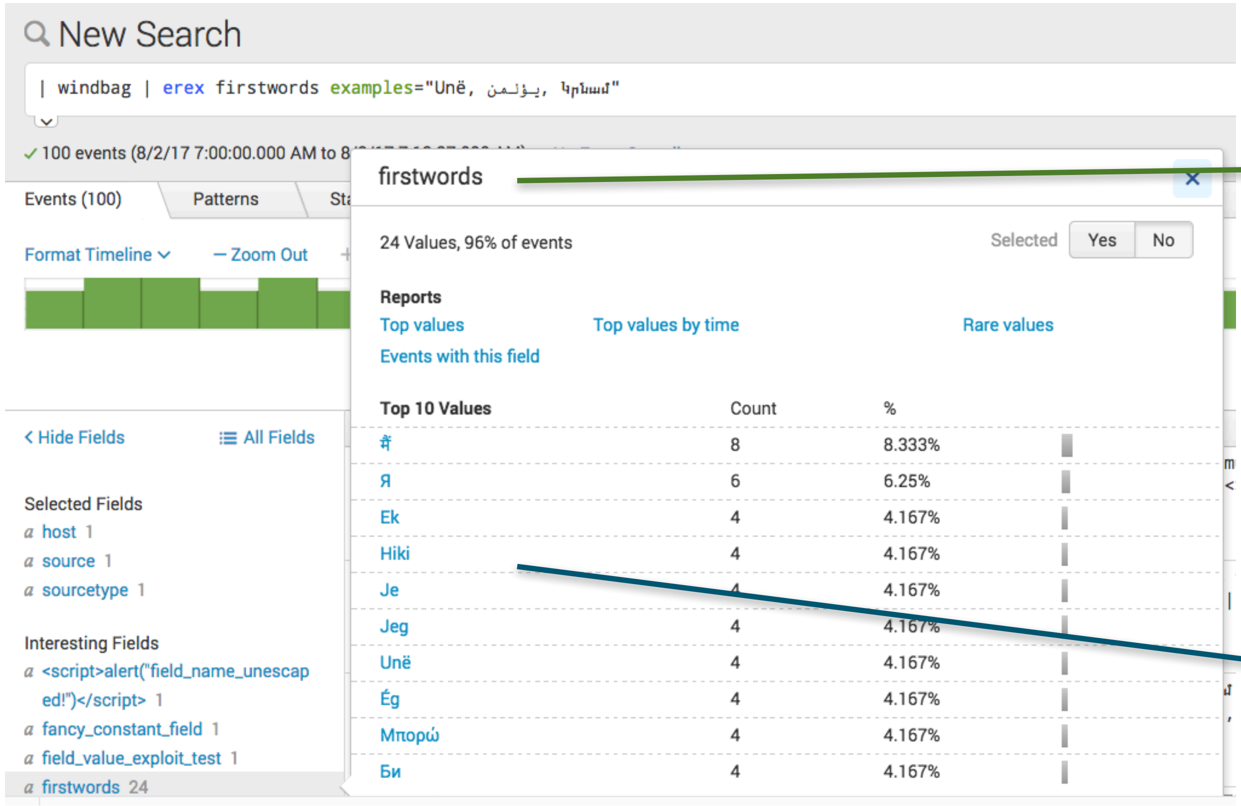
Easter egg that creates sample data

New Field to create

Examples from the data

erex Command

Field Extractions Using Examples



New Search

```
| windbag | erex firstwords examples="Unë, يؤلمن, Կրնամ"
```

✓ 100 events (8/2/17 7:00:00.000 AM to 8

Events (100) | Patterns | Sta

Format Timeline — Zoom Out

< Hide Fields   ☰ All Fields

Selected Fields
- a host 1
- a source 1
- a sourcetype 1

Interesting Fields
- a <script>alert("field_name_unescaped!")</script> 1
- a fancy_constant_field 1
- a field_value_exploit_test 1
- a firstwords 24

firstwords

24 Values, 96% of events — Selected: Yes | No

Reports
Top values | Top values by time | Rare values
Events with this field

Top 10 Values	Count	%
मैं	8	8.333%
Я	6	6.25%
Ek	4	4.167%
Hiki	4	4.167%
Je	4	4.167%
Jeg	4	4.167%
Unë	4	4.167%
Ég	4	4.167%
Μπορώ	4	4.167%
Би	4	4.167%

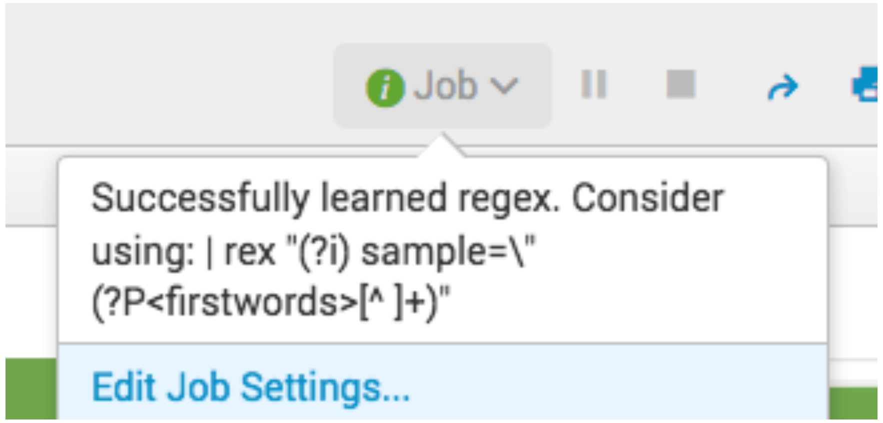
New Field created

| windbag | erex firstwords examples="Unë, يؤلمن, Կրնամ"

The values erex generated based on the samples

erex Command

Field Extractions Using Examples



Successfully learned regex. Consider using: | rex "(?i) sample=\"(?P<firstwords>[^]+)"

- Erex is a great introduction to using regular expressions for field extraction.
  - Erex provides the rex that it generated
  - Going forward, use the rex in your saved searches and dashboards.
  - Rex is more efficient

rex Command

Extract Fields Using Regular Expressions at Search Time

Creates a Field Extraction

```
… | rex field={what_field} “FrontAnchor(?<extraction>{characters}+)BackAnchor”
```

rex Command

Extract Fields Using Regular Expressions at Search Time

```
| windbag | rex field=sample "^(?<FirstWord>[\S+]*)"
```

Specify the field to rex from

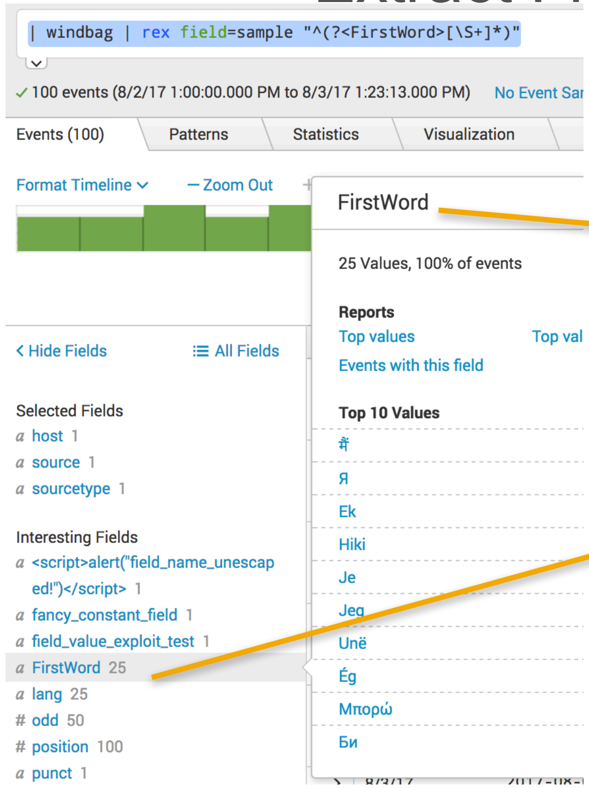
Front Anchor

Named Field Extraction

Grab any non-space character

rex Command

Extract Fields Using Regular Expressions at Search Time



```
| windbag | rex field=sample "^(?<FirstWord>[\S+]*)"
```

✓ 100 events (8/2/17 1:00:00.000 PM to 8/3/17 1:23:13.000 PM)   No Event Sa

Events (100) | Patterns | Statistics | Visualization

Format Timeline   — Zoom Out

< Hide Fields   All Fields

Selected Fields
- a host 1
- a source 1
- a sourcetype 1

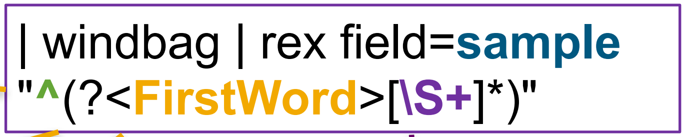
Interesting Fields
- a <script>alert("field_name_unescaped!")</script> 1
- a fancy_constant_field 1
- a field_value_exploit_test 1
- a FirstWord 25
- a lang 25
- # odd 50
- # position 100
- a punct 1

FirstWord

25 Values, 100% of events

Reports
Top values   Top val
Events with this field

Top 10 Values
- मैं
- Я
- Ek
- Hiki
- Je
- Jeg
- Unë
- Ég
- Μπορώ
- Би



| windbag | rex field=**sample**
"**^**(?<**FirstWord**>[**\S+**]*)"

Named Field Extraction

Grab any non-space character

rex Command

Use Rex to Perform SED Style Substitutions

SED is a stream editor. It can be used to create substitutions in data.

Splunk uses the rex command to perform Search-Time substitutions.

rex Command

Use Rex to Perform SED Style Substitutions

```
| windbag | search lang="*Norse"
| rex mode=sed "s/Old (Norse)/Not-so-old \1/g"
```

Set the mode

s for substitute

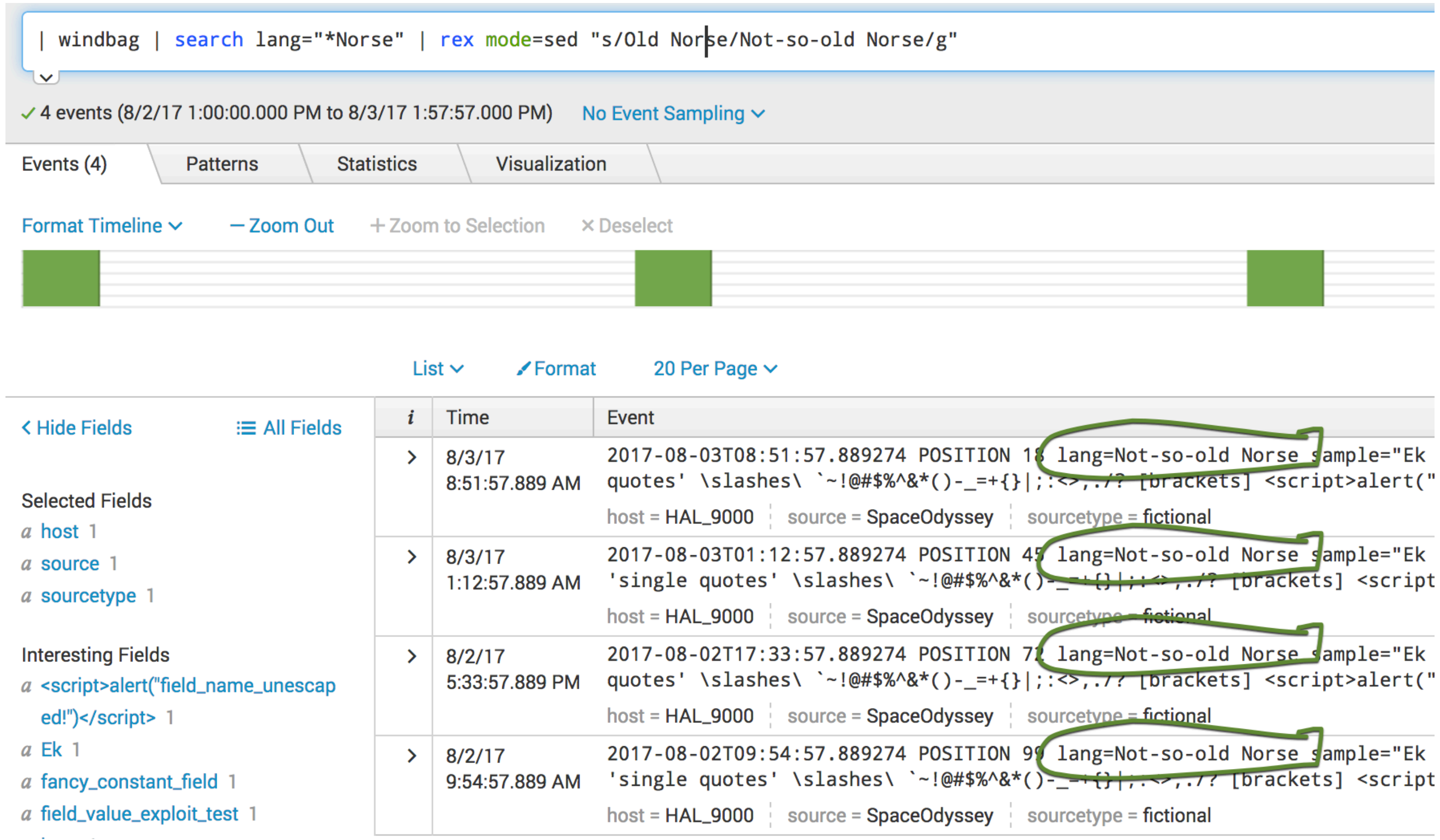
() to create a capture group
\1 to paste capture group

Substitute the stuff between the first / and second / with the stuff between second / and third /

g for global
(more than once)

rex Command

Use Rex to Perform SED Style Substitutions



```
| windbag | search lang="*Norse" | rex mode=sed "s/Old Norse/Not-so-old Norse/g"
```

✓ 4 events (8/2/17 1:00:00.000 PM to 8/3/17 1:57:57.000 PM) No Event Sampling

Events (4) | Patterns | Statistics | Visualization

Format Timeline | — Zoom Out | + Zoom to Selection | × Deselect

List | Format | 20 Per Page

< Hide Fields | All Fields

Selected Fields
- a host 1
- a source 1
- a sourcetype 1

Interesting Fields
- a <script>alert("field_name_unescaped!")</script> 1
- a Ek 1
- a fancy_constant_field 1
- a field_value_exploit_test 1

i	Time	Event
>	8/3/17 8:51:57.889 AM	2017-08-03T08:51:57.889274 POSITION 18 lang=Not-so-old Norse sample="Ek quotes' \slashes\ `~!@#$%^&*()-_=+{}|;:<>,./? [brackets] <script>alert(" host = HAL_9000 source = SpaceOdyssey sourcetype = fictional
>	8/3/17 1:12:57.889 AM	2017-08-03T01:12:57.889274 POSITION 45 lang=Not-so-old Norse sample="Ek 'single quotes' \slashes\ `~!@#$%^&*()-_=+{}|;:<>,./? [brackets] <script host = HAL_9000 source = SpaceOdyssey sourcetype = fictional
>	8/2/17 5:33:57.889 PM	2017-08-02T17:33:57.889274 POSITION 72 lang=Not-so-old Norse sample="Ek quotes' \slashes\ `~!@#$%^&*()-_=+{}|;:<>,./? [brackets] <script>alert(" host = HAL_9000 source = SpaceOdyssey sourcetype = fictional
>	8/2/17 9:54:57.889 AM	2017-08-02T09:54:57.889274 POSITION 99 lang=Not-so-old Norse sample="Ek 'single quotes' \slashes\ `~!@#$%^&*()-_=+{}|;:<>,./? [brackets] <script host = HAL_9000 source = SpaceOdyssey sourcetype = fictional

Result:

…
| rex mode=sed "s/Old (Norse)/Not-so-old \1/g"

Evaluation

Using Regular Expressions for Pattern Matching

Regex Command

Filter Using Regular Expressions

```
sourcetype=fs_notification | regex chgs="^modtime"
```

Field to evaluate

Regex

Match Function

Filter Using Regular Expressions

```
match(SUBJECT),”REGEX”

    … | eval n = if(match(field,”^MyRegex”, 1, 2)

sourcetype=access_combined_wcookie
| eval com = if(match(referer,"http:.*\.com"),"True","False")
```

Match. Returns 1 for it matches, 0 for not.

Field to evaluate

The Regex

Replace Command

Switch Data at Search Time

Replace field values with the values you specify

```
… | replace “<whoever>” WITH “<whomever>” IN <target_field>
```

Replace Command

Switch Data at Search Time

Replace field values with the values you specify

… | replace “<whoever>” WITH “<whomever>” IN <target_field>

| windbag | replace "**Euro**" **with** "**Euro: How is a currency a language**" **in** **lang**

String to be replaced

operator

String to replace with

operator

Field in which to make the replacement

splunk> .conf2017

Persistence

Regular Expressions That Exist Outside Your Search

Until this point, every one of our extractions have only existed in the search. But, what if we want them to persist? Or to share them?

1. Interactive Field Extractor

2. Extractions in Props / Transforms

Persistent Field Extractions

Comparing The Persistent Field Extractions

Interactive Field Extractor

- *Walk-through UI*
- *You may want to rewrite the generated Regex*
- *Does not require admin rights*

Extract in Props

- *Straight editing in props.conf*
- *Requires Admin Rights (or an admin to put in place)*

Report in Transforms

- *Edit directly in transforms.conf*
- *Invoked by props.conf*
- *Requires Admin Rights (or an admin to put in place)*

Q&A

Michael Simko | Sr. Engineer/Instructor

splunk> .conf2017

Key Takeaways

Regex in your SPL

1. Use Regex to create powerful filters in your SPL
2. Use Regex to create field extractions
3. Regex doesn't have to be hard. You can do this!

splunk> .conf2017

Thank You

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

splunk> .conf2017

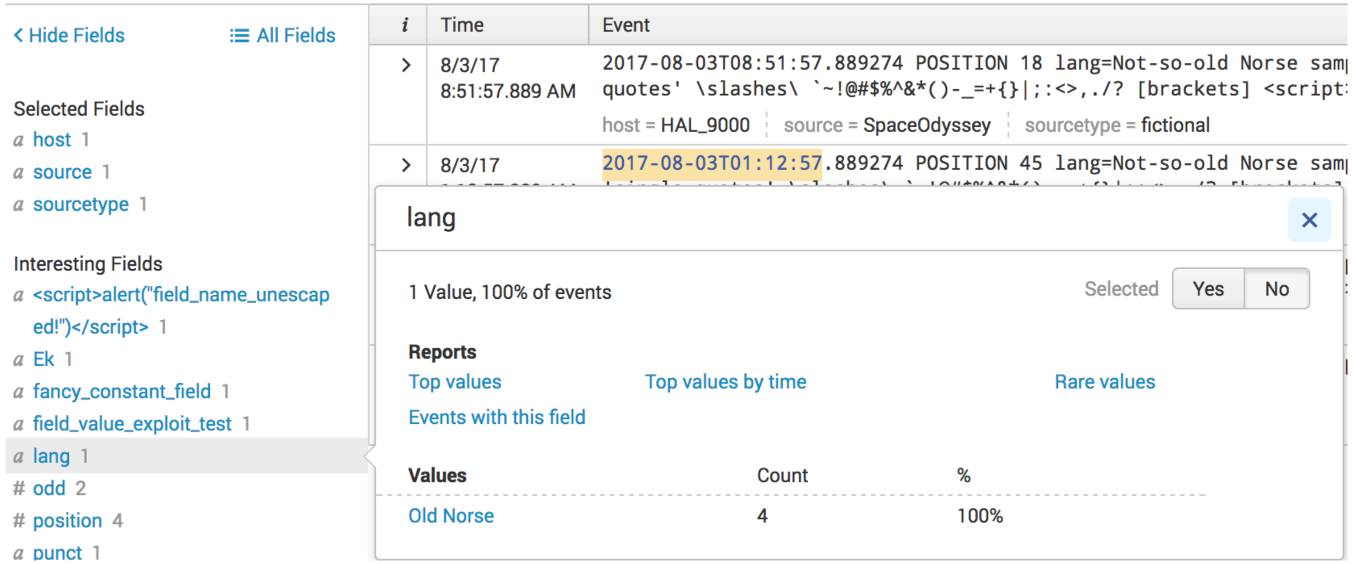
Appendix A

Caveats

rex Command – Caveat

Use Rex to Perform SED Style Substitutions

```
| windbag | search lang="*Norse"
| rex mode=sed "s/Old (Norse)/Not-so-old \1/g"
```



< Hide Fields   ☰ All Fields

Selected Fields
- a host 1
- a source 1
- a sourcetype 1

Interesting Fields
- a <script>alert("field_name_unescaped!")</script> 1
- a Ek 1
- a fancy_constant_field 1
- a field_value_exploit_test 1
- a lang 1
- # odd 2
- # position 4
- a punct 1

i	Time	Event
>	8/3/17 8:51:57.889 AM	2017-08-03T08:51:57.889274 POSITION 18 lang=Not-so-old Norse sam quotes' \slashes\ `~!@#$%^&*()-_=+{}|;:<>,./? [brackets] <script host = HAL_9000 ¦ source = SpaceOdyssey ¦ sourcetype = fictional
>	8/3/17	2017-08-03T01:12:57.889274 POSITION 45 lang=Not-so-old Norse sam

lang ✕

1 Value, 100% of events   Selected [✓ Yes] No

Reports
Top values   Top values by time   Rare values
Events with this field

Values	Count	%
Old Norse	4	100%

Caveat:

The substitution from rex comes after the lang field is extracted.
So even though the event data is showing us the substitution, the field lang is showing the original value.

splunk> .conf2017

Appendix B

Exercises to Practice With

Regex Basics

The Main Elements

Control Characters:

^ Start of a Line
$ End of a Line

Character Types:

\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Operators:

* Zero or More
+ One or More
? Zero or One

Scenario Regex: ^\d+\s\w+\d+\s\d+:\d+:\d+

Learn by Fire:
Which of these will the sample Regex match?

A. 002421 Februari 1083 1:242525:22352
B. 07 Feb 17 12:53:36AM
C. Feb 13 2017 18:46:56
D. 14 February 2017 07:45:47Z

(answers on next slide)

splunk> .conf2017

Regex Basics

The Main Elements

Control Characters:

^ Start of a Line
$ End of a Line

Character Types:

\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Operators:

* Zero or More
+ One or More
? Zero or One

Scenario Regex: ^\d+\s\w+\d+\s\d+:\d+:\d+

Learn by Fire:
Which of these will the sample Regex match?

A. 002421 Februari 1083 1:242525:22352
B. 07 Feb 17 12:53:36AM
C. Feb 13 2017 18:46:56
D. 14 February 2017 07:45:47:46

Regex doesn't care if it looks wrong,
It only cares if it matches the pattern

Regex Basics

The Main Elements

Control Characters:

^ Start of a Line
$ End of a Line

Character Types:

\s White Space
\S Not white space
\d Digit
\D Not Digit
\w Word Character
\W Not Word Characters

Operators:

* Zero or More
+ One or More
? Zero or One

Practice: Create **a** Regex that describes all three of the following strings

```
06 February 2017 192.168.1.2
05 Apr 2014 10.2.1.150
31 July 2020 19..15.63
```

Regex Basics

The Main Elements

Control Characters:

^ Start of a Line

$ End of a Line

Character Types:

\s White Space

\S Not white space

\d Digit

\D Not Digit

\w Word Character

\W Not Word Characters

Operators:

* Zero or More

+ One or More

? Zero or One

Scenario: Create a Regex that describes the following strings

A solution:

```
\d+\s\w+\s\d+\s\d*\.\d*\.\d*\.\d*
```

06 February 2017 192.168.1.2

05 Apr 2014 10.2.1.150

31 July 2020 19..15.63

Regex Basics

The Main Elements

1. Open up your Splunk
2. | windbag | head 20 | table _raw
3. Copy the _raw data
4. Paste the data in Regex101.com

Goals: Extract the following fields for each event:
lang
sample
The Date without Time
The Time

Perform these as “named” extractions

Replace Command

Switch Data at Search Time

Silly version to try on your own

```
| windbag | head 20 | replace "1" WITH "Uno" in odd
```

Try it, then click the down chevron to see the results