Beyond “Regular” Regular Expressions

Cary Petterborg | Splunk Architect | LDS Church

August 8, 2017
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.
My Disclaimer

During the course of this presentation, I may make references to my employer, The Church of Jesus Christ of Latter-day Saints. This should not be taken as an endorsement of Splunk or Splunk products by the LDS Church.
Who is Cary Petterborg

- Splunk user and administrator for 5.5 years
- Monitoring Engineer for 10 years
- Web developer for 23 years
- Software engineer for 37 years
- Many languages from assembly to Ruby
- Application development including Flight Sim, DB systems, and Web
- Works for the LDS Church in Salt Lake City
- Speaker at .conf 201[567]
- SplunkTrust Member 2018
My purpose here today is to...

- Help you **control your data** instead of letting your data control you

- Regular expressions give you that control
Where’s Waldo?
Can you easily pick out all the female Waldo’s?

Picture courtesy of Albanpix.com
Regular Expressions help you…

- Find the distinctions within similar data
- Isolate the value properties from the noise
- Find the needles in the haystacks
- Break data into usable, constituent parts
Why Do I Like Regular Expressions?

- Using Regular Expressions since the mid 80’s
- Started using regex with lex/yacc/sed/grep for software development
- Realized the power of regex quickly
- Taught classes on regex
- Love working with regex stuff in Splunk and other utilities
- Regex is an important skill, and I want to share my knowledge
- Have Rex – Will Conquer
One day, you too...
One day, you too...
Splunk and Regular Expressions
Where do you use regular expression in Splunk?

- Field extractions
- The rex and regex search commands
- In props.conf, transforms.conf and other .conf files
- Data feeds (probably external to Splunk itself)
- Note: Splunk regular expressions are PCRE (Perl Compatible Regular Expressions) and use the PCRE C library.
Splunk Field Extraction Tool
GUI tool in the web UI of Splunk

Simple to use, and you can visually see the results of the regex on the events

Pretty good *for a start*, but not *always* good for a final result

Not able to optimize or do anything complex

Makes mistakes if you don’t have regular data
Example of why you might want to use your own regex for field extraction

Or, how to be smarter than the Field Extraction Tool
<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>User</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon Jun 06 2016 02:29:19</td>
<td>www1 sshd[3849]</td>
<td>Failed password for root from 128.241.220.82 port 2253 ssh2&lt;</td>
<td></td>
</tr>
<tr>
<td>Mon Jun 06 2016 02:29:24</td>
<td>www1 sshd[4267]</td>
<td>Failed password for invalid user administrator from 128.241.220.82 port 1715 ssh2&lt;</td>
<td></td>
</tr>
<tr>
<td>Mon Jun 06 2016 02:29:44</td>
<td>www1 sshd[5001]</td>
<td>Failed password for invalid user george from 128.241.220.82 port 2212 ssh2&lt;</td>
<td></td>
</tr>
<tr>
<td>Mon Jun 06 2016 02:30:13</td>
<td>www1 sshd[1638]</td>
<td>pam_unix(sshd:session): session opened for user djohnson by (uid=0)&lt;</td>
<td></td>
</tr>
<tr>
<td>Mon Jun 06 2016 02:30:13</td>
<td>www1 sshd[1480]</td>
<td>Failed password for invalid user yp from 128.241.220.82 port 2808 ssh2&lt;</td>
<td></td>
</tr>
<tr>
<td>Mon Jun 06 2016 02:30:22</td>
<td>www1 sshd[2291]</td>
<td>Failed password for invalid user email from 128.241.220.82 port 4995 ssh2&lt;</td>
<td></td>
</tr>
<tr>
<td>Mon Jun 06 2016 02:30:26</td>
<td>www1 sshd[4761]</td>
<td>Failed password for invalid user local from 128.241.220.82 port 1271 ssh2&lt;</td>
<td></td>
</tr>
<tr>
<td>Mon Jun 06 2016 02:30:50</td>
<td>www1 sshd[4906]</td>
<td>Failed password for invalid user mysql from 128.241.220.82 port 2075 ssh2&lt;</td>
<td></td>
</tr>
<tr>
<td>Mon Jun 06 2016 02:31:19</td>
<td>www1 sshd[81145]</td>
<td>pam_unix(sshd:session): session opened for user djohnson by (uid=0)&lt;</td>
<td></td>
</tr>
<tr>
<td>Mon Jun 06 2016 02:31:19</td>
<td>www1 sshd[2021]</td>
<td>Failed password for myuay from 10.1.10.172 port 4468 ssh2&lt;</td>
<td></td>
</tr>
<tr>
<td>Mon Jun 06 2016 02:31:28</td>
<td>www1 sshd[1155]</td>
<td>Failed password for invalid user yp from 10.1.10.172 port 1822 ssh2&lt;</td>
<td></td>
</tr>
<tr>
<td>Mon Jun 06 2016 02:31:32</td>
<td>www1 sshd[1632]</td>
<td>Failed password for invalid user elena_endubasquet from 10.1.10.172 port 2074 ssh2&lt;</td>
<td></td>
</tr>
<tr>
<td>Mon Jun 06 2016 02:31:54</td>
<td>www1 sshd[4333]</td>
<td>Failed password for invalid user root from 10.1.10.172 port 2772 ssh2&lt;</td>
<td></td>
</tr>
<tr>
<td>Mon Jun 06 2016 02:32:00</td>
<td>www1 sudo: djohnson ; TTY=pts/0 ; PWD=/home/djohnson ; USER=root ; COMMAND=/bin/su</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mon Jun 06 2016 02:32:00</td>
<td>www1 sshd[3697]</td>
<td>Failed password for invalid user whois from 10.1.10.172 port 1246 ssh2&lt;</td>
<td></td>
</tr>
<tr>
<td>Mon Jun 06 2016 02:32:19</td>
<td>www1 sshd[5985]</td>
<td>Failed password for invalid user testuser from 10.1.10.172 port 2597 ssh2&lt;</td>
<td></td>
</tr>
</tbody>
</table>
Simple FET extraction of Port

This is what the FET gives you:

```
Select Fields
Highlight one or more values in the sample event to create fields. You can indicate one value is required, meaning it must exist in an event for the regular expression.

Wed Jun 08 2016 09:39:12 www sshd[4092]: Failed password for invalid user local from 109.169.32.135 port 2172 ssh2

Hide Regular Expression

^(?::[^\n]*[^\n])\d+\s+\w+\s+(?P<port>\d+)$
```
Intelligent extraction of Port

Regular Expression

port\s+\(?P<port>\d+\)
Notes on Named Capture Groups

- (?P<name>…) === (?<name>…)
- The P is optional (came from Python), but it is usually considered more correct
- Splunk FET will use (?P<name>…), so why not make things similar?

**BUT**

- Do it the way you feel most comfortable
Goal: user from all entries using one regex
There is no user automatically extracted
FET Failed
Gets wrong values from some events

- Failed password for invalid user **local** from 109.169.32.135 port 2
- Failed password for nsharpe from **10.2.10.163** port 8317 ssh2
- Failed password for invalid user **dean** from 109.169.32.135 port 38
- Failed password for invalid user **operator** from 109.169.32.135 port
- Failed password for invalid user **itmadmin** from 109.169.32.135 port
- Failed password for mail from **84.34.159.23** port 1190 ssh2
- Failed password for sync from **84.34.159.23** port 2530 ssh2
- Failed password for invalid user **inet** from 10.3.10.46 port 1516
- ]: pam_unix(sshd:session): session closed for user myuan by (uid=0)
- Failed password for invalid user **administrator** from 10.3.10.46 port
FET Failed After More Lines Added
Trying to add an additional line and extracting user doesn’t work

⚠️ The extraction failed. If you are extracting multiple fields, try removing one or more fields. Start with extractions that are embedded within longer text strings.

Select Fields
Highlight one or more values in the sample event to create fields. You can indicate one value is required, meaning it must exist in an event for the regular expression to match. Click on highlighted values.

Wed Jun 08 2016 09:39:12 wv1 sshd[4092]: Failed password for invalid user local from 109.169.32.135 port 2172 ssh2

Wed Jun 08 2016 09:39:12 wv1 sshd[38618]: Failed password for msharpe from 10.2.163 port 8317 ssh2

Preview
If you see incorrect results below, click an additional event to add it to the set of sample events. Highlight its values to improve the extraction. You can remove incorrect values in the next step.
Let’s look at the generated REGEX
Not pretty, not easy to change/fix, not efficient. So, let’s fix it.
My tool of choice: regex101.com
Add the data and a regex

```regex
/for (?P<user>\S+) from
```

### TEST STRING

<table>
<thead>
<tr>
<th>Time</th>
<th>Host</th>
<th>Process</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon Jun 06 02:29:19</td>
<td>www1</td>
<td>sshd[3849]:</td>
<td>Failed password for root from 128.2</td>
</tr>
<tr>
<td>Mon Jun 06 02:29:24</td>
<td>www1</td>
<td>sshd[4267]:</td>
<td>Failed password for invalid user admin</td>
</tr>
<tr>
<td>Mon Jun 06 02:29:44</td>
<td>www1</td>
<td>sshd[5001]:</td>
<td>Failed password for invalid user generated</td>
</tr>
<tr>
<td>Mon Jun 06 02:30:13</td>
<td>www1</td>
<td>sshd[1638]:</td>
<td>pam_unix(sshd:session): session opened</td>
</tr>
<tr>
<td>Mon Jun 06 02:30:13</td>
<td>www1</td>
<td>sshd[1480]:</td>
<td>Failed password for invalid user yp</td>
</tr>
<tr>
<td>Mon Jun 06 02:30:22</td>
<td>www1</td>
<td>sshd[2291]:</td>
<td>Failed password for invalid user example</td>
</tr>
<tr>
<td>Mon Jun 06 02:30:26</td>
<td>www1</td>
<td>sshd[4761]:</td>
<td>Failed password for invalid user local</td>
</tr>
<tr>
<td>Mon Jun 06 02:30:50</td>
<td>www1</td>
<td>sshd[4986]:</td>
<td>Failed password for invalid user my</td>
</tr>
<tr>
<td>Mon Jun 06 02:31:19</td>
<td>www1</td>
<td>sshd[81145]:</td>
<td>pam_unix(sshd:session): session opened</td>
</tr>
<tr>
<td>Mon Jun 06 02:31:19</td>
<td>www1</td>
<td>sshd[2021]:</td>
<td>Failed password for myuan from 10.1</td>
</tr>
<tr>
<td>Mon Jun 06 02:31:28</td>
<td>www1</td>
<td>sshd[1155]:</td>
<td>Failed password for invalid user yp</td>
</tr>
<tr>
<td>Mon Jun 06 02:29:19</td>
<td>www1</td>
<td>sshd[3849]:</td>
<td>Failed password for root from 128.2</td>
</tr>
<tr>
<td>Mon Jun 06 02:29:24</td>
<td>www1</td>
<td>sshd[4267]:</td>
<td>Failed password for invalid user admin</td>
</tr>
</tbody>
</table>
Refine the regex – better matches, but not all
Refine the regex again – almost there

REGULAR EXPRESSION

```regex
!/^ for (.*invalid user )?(.*user .*)?\S+ (.*) \{|From\|by\}/
```
And FINALLY – we got them all
Four different formats – all four user field types found!

<table>
<thead>
<tr>
<th>Regular Expression</th>
<th>Test String</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[:]](\(\?if\(invalid\ user\ D][\(user\ D]?))))(?P&lt;user&gt;\S+)\(from\)\(by\)</code></td>
<td>Mon Jun 06 2016 02:29:19 www1 sshd[3849]: Failed password for root from 128.241.220.82 port 2253 ssh2</td>
</tr>
</tbody>
</table>
Back to the FET – use our regex

Paste this regex into the field that the ugly, single-format regex was in:

```
((for ((invalid user ))(user ))?)(sudo: ))(?P<user>\S+) (from|by)?
```
...and the results are MUCH better
How’d He Do That?

Tricks for getting it right
One named capture group with a single name

More than one instance of the same name will fail

- (for invalid user (?P<user>\S+))\| (for (?P<user>\S+))

- Capture group names must be unique:
How do you eat an elephant?

Bite by bite is better than trying to stuff the whole elephant in your mouth at once

- Start with one format
- Try to find similarities and differences between the formats
- Add a new format to your data and check your updated regex
- Keep a copy of the last one that worked!!
- Add additional formats and check ALL matches for ALL examples
Alternate: Do one for each format

Then you can try combining

- This can be more difficult with simpler regexes

- Can be easier for more complex regexes

- Combine two of the regexes that are similar

- Try to keep the things that are the same in both, making the changes to the original only where there is a difference

- Remember – **ONE instance of a name per regex**
Use Parentheses!
Make it easier to come back to later

((for ((invalid user )||(user )))?(sudo: ))(?P<user>\S+) (from|by)?

▶ Using parentheses for clarity is helpful:

▶ They make it possible to see the separate parts and their relationships with each other

▶ Don’t overdo the parentheses
Use the Best Character Class
Use the right tool for the job

- Sometimes a field regex must be able to match data that hasn’t been seen in the data yet, so in this case **be as general as possible**

- In the previous example, the `\S` is best because `\s` will be the delimiter (a space in this case) because you want to catch any potential case that you don’t see in the data, *yet*.

  `\S+`

- If you have a delimiter that you can count on, use something like this to match the field value (**in this case be specific** about what it is NOT):

  `[\^,]+`
Just because your sample data doesn’t have a particular character in it, that doesn’t mean it never will. Examples:

▶ Usernames – alphanumerics + what?
  • Dash – Underscore - Other characters - Are you sure?

▶ Filenames – could you have a space?
  • C:\Program Files\My Application\
If you have a definite delimiter, take advantage

Examples:

▶ Data: "contents of quoted string"
  • Use: "(?P<contents>[^"]+)"

▶ Data: User:carypetterborg Dept:ICS
  • Use: User:(?P<user>\S+)\s

▶ Data: Salt Lake City, Utah 84117-6403
  • Use: ^(?P<city>[^,]+),\s+(?P<state>.+)\s+(?P<zip>[-\d]*)$
REX and REGEX Commands

The most common use for regular expressions is in SPL with `rex` and `regex`
When to Use REX

When you:

▶ don’t always want to extract the data
▶ want to extract data from a field that is already extracted
▶ don’t have access to field extractions (permissions, etc.)
▶ require doing multiple, disparate regular expressions
▶ are in a hurry or you are doing a proof-of-concept
REX Example

```plaintext
index=voice sourcetype=voice* Description | rex "Description=(?P<description>\[^\^\]+)"
| rex field=description "From (?P<start>.+) to (?P<end>.+?):\s"
```

REX Commands

It takes two:

- `index=voice sourcetype=voice`  
  
  | rex "Description=(?P<description>[^\^]+)"
  | rex field=description "From (?P<start>.+) to (?P<end>.+?):\s"

SYNTAX:

- rex [field=fieldname] “regex”

Also available:

- rex mode=ed
First rex – get the description

REGULAR EXPRESSION

/:\bDescription=\(?P<description>[^\s]+\)\b/ g

TEST STRING

Second rex – get the Start and End

Number of AuthenticationFailed events exceeds configured threshold during configured interval of time 1 within 3 minutes on cluster StandAloneCluster. There are 2 AuthenticationFailed events (up to 30) received during the monitoring interval From Wed Aug 03 10:25:00 PHT 2016 to Wed Aug 03 10:28:00 PHT 2016: TimeStamp: 8/3/16 10:26 AM
When to use REGEX

- To filter out events/data that you don’t want included in the pipeline
- This is like **search** on steroids, but doesn’t replace **search**
- Only used as a filter
**Regex example**

Only get events with internal addresses
Search: `sourcetype=linux_secure | regex "10\d+\d+\d+\d+"`

Only internal (10.*) IP addresses make it through the regex filter

Search produces events, regex then limits those results passed on through the pipeline by a fancy regular expression

Yes, there are other ways to do this, but this is a regex example
Rex vs Regex

- Use **rex** to extract fields
- Use **regex** to limit results
- Yes, you can use them in the same search:

  ```
  sourcetype=linux_secure | rex "from (?P<src_ip>\d+\.\d+\.\d+\.\d+)") | regex src_ip="(?<!10)\.\d+\.\d+\.\d+
  ```
Index Time Regular Expression Usage
The Problem

- You can’t index Social Security Numbers

- How do you distinguish a Social Security Number from other numbers?

- Obfuscate ONLY SSNs, but leave other things alone.
SSN vs Phone#

Regex distinctions

SSN

- 123-45-6789
- \d{3}-\d{2}-\d{4}

Phone #

- 800-123-4567
- \d{3}-\d{3}-\d{4}
Be as specific in your matches as possible

- You could use something simple like:
  \d+-\d+-\d+

- But it will mistake a phone number for a SSN:
A Better Match

- New regex:

```regex\d+-\d\d-\d+```

- Gets rid of Phone #’s, but what about other data?
We’re now so close

- This is exactly what a properly formatted SSN looks like:

  \d{3}-\d{2}-\d{4}

- This defines a SSN, but it matches other things, too:

  ```
  REGULAR EXPRESSION
  :
  (\d{3}-\d{2}-\d{4})
  
  TEST STRING
  
  My phone number is 800-123-4567
  My SSN is 123-45-6789
  My birthday is 12-25-1960
  I need part # 8871-34-91268
  ```
Let’s get REAL specific

- Best definition:
  
  ```
  (?<!\d)(?P<ssn>\d{3}-\d{2}-\d{4})(?!\d)
  ```

- The SSN match can be found *anywhere* in the event, and only the SSN:
So let’s make it work in transforms.conf

- Grab the beginning and ending text of the event:

  \((.*)(?<!\d)(\d{3}-\d{2}-\d{4})(?!\d)(.*)\)
Index-time conversion

Conditions and Limitations

- This regex can’t be done with SEDCMD in props.conf alone
  - The regex uses regex features not found in SED format

- Using a simple custom sourcetype, but it can be made a general transform

- Must capture all parts of the event.

- Will obfuscate only one SSN per event.
[testssn]
TRANSFORM-ssn = nossn
DATETIME_CONFIG = CURRENT
NO_BINARY_CHECK = true
SHOULD_LINEMERGE = false
disabled = false
The transforms.conf

[nossn]
REGEX=(?m)(.*)(?<!\d)(\d{3}-\d{2}-\d{4})(?!\d)(.*)
FORMAT = $1###-##-#####$3
DEST_KEY = _raw
New regex features shown

- `(?:m)` – perform the regex on multi-line events

- `(?!\d)` - Negative Lookbehind – *not preceeded by a digit* – no net character

- `(?<!\d)` – Negative Lookahead – *not followed by a digit* – no net character
How the FORMAT works

- FORMAT = $1###-##-####$3

- $1$ and $3$ are capture group matches – from (.* ) at beginning and end

- $2$ is not used in the FORMAT, but it’s the capture group for the SSN – from: (\d\{3\}-\d\{2\}-\d\{4\})
After bringing in the data:

```
<table>
<thead>
<tr>
<th>i</th>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7/18/17 2:16:51.000 PM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7/18/17 2:16:51.000 PM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7/18/17 2:16:51.000 PM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7/18/17 2:16:51.000 PM</td>
<td></td>
</tr>
</tbody>
</table>

**host** = Carys-MacBook-Pro-2.local
**source** = ssn.log
**sourcetype** = testssn
```
It even will obfuscate the data import preview:

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/18/17</td>
<td>My phone number is 800-123-4567. timestamp = none</td>
</tr>
<tr>
<td></td>
<td>My SSN is ###-###-####. timestamp = none</td>
</tr>
<tr>
<td>7/18/17</td>
<td>My birthday is 12-25-1960. timestamp = none</td>
</tr>
<tr>
<td>7/18/17</td>
<td>I need part # 8871-34-91268. timestamp = none</td>
</tr>
<tr>
<td></td>
<td>###-###-####</td>
</tr>
</tbody>
</table>

**Source type:** testssn

**Break Type:** Auto, Every Line, Regex...

**Timestamp**

**Extraction:** Auto, Current time, Advanced...
Greedy vs. Lazy Matches
Greedy vs Lazy
What is the difference?
Subtle difference, but big effect

▶ **Greedy** – Grab as much as you can
▶ **Lazy** – Grab as little as you can

▶ The lazy match will continue only as far as it needs to, no further
  • `<.+?>` will match `<12345>`, while
  • `<.+>` will match both `<12345>` and `<12345><67890>`

SYNTAX: place a `?` After a `*` or `+`

The lazy match only goes to the first instance of a match following a multiple match
Greedy Example

REGULAR EXPRESSION

```
/\([^P<cmd>\.*\)session\]/
```

```
led password for invalid user geo
  _unix(sshd:session): session  open
  led password for invalid user yp
  led password for invalid user ema
  led password for invalid user loc
  led password for invalid user mys
  _unix(sshd:session): session  open
  led password for myuan from 10.1.
```
Lazy Example

```regex
!/\((?P<cmd>.*?)\)session
```

```
...ed password for invalid user ge
...ed password for invalid user ge
...ed password for invalid user ge
...ed password for invalid user ge
...ed password for invalid user ge
...ed password for invalid user my
...```

```bash
splunk> .conf2017
```
Second Look - Greedy

REGULAR EXPRESSION

/\ From \(?P\<start\>.+\) to \(?P\<end\>.+\):\s\</g |

TEST STRING

Number of AuthenticationFailed events exceeds configured threshold during configured interval of time 1 within 3 minutes on cluster StandAloneCluster. There are 2 AuthenticationFailed events (up to 30) received during the monitoring interval From Wed Aug 03 10:25:00 PHT 2016 to Wed Aug 03 10:28:00

Second Look - Lazy

```regex
/From (?P<start>.*+) to (?P<end>.*?):\s*/g
```

**TEST STRING**

Number of AuthenticationFailed events exceeds configured threshold during configured interval of time 1 within 3 minutes on cluster StandAloneCluster. There are 2 AuthenticationFailed events (up to 30) received during the monitoring interval From Wed Aug 03 10:25:00 PHT 2016 to Wed Aug 03 10:28:00 PHT 2016

**PHT 2016:**

- TimeStamp: 8/3/16 10:26 AM
- LoginFrom: 172.12.34.40
- Interface: VMREST
- UserID: JacobMD
- AppID: Cisco Tomcat
- ClusterID: NodeID: APPHMANAOVM001
- TimeStam::Status=2,cleared
Choose Wisely

➤ Greedy may cross long segments

➤ Lazy may stop prematurely

➤ **Try it on various data sets** to make sure it will do what you want
Embedding
Multiple field extractions from one piece of data

► Problem:
  • Extract two different fields from the exact same piece of data
  • Only want to use one regex – for efficiency if nothing else
  • Need both the Domain only and the whole URL from an access log
Source and Results

Data:
• 1501408932.060 16922 108.65.113.83 TCP_REFRESH_HIT/200 474 GET http://damtare.by.ru/id.txt myuan@buttercupgames.com DIRECT/damtare.by.ru text/html DEFAULT_CASE-DefaultGroup-Demo_Clients-NONE-NONE-DefaultRouting <IW_scty,-6.9,0,-,- ,,-,0,-,-,-,-,-,-,-,-,-,IW_scty,-> - -

Desired Fields:
• Domain: damtare.by.ru
• URL: http://damtare.by.ru/id.txt
Regex
Regex101.com vs Splunk

- Slashes need escaping in regex101, but not in Splunk:

```
(?:P<URL>http:W(?:P<domain>[^V]+)S+)
```

vs

```
(?:P<URL>http:\/(?:P<domain>[^/]+)S+)
```
What? ... Oh, now I see.

(?P<URL>http://(?P<domain>[^/]+)\S+)
Results in Splunk
You can’t do this in the FET without doing your own regex!

```
(?P<URL>http://(?P<domain>(\./)+)\S+)
```
Performance Considerations

- Some complex field extractions can be costly

- Some complex regular expressions can be costly

- **Use the Job Inspector** to see if there is a difference in doing on complex field extraction vs man simple field extractions (rex vs rex rex rex)

- **Sometimes the readability is more important** than the performance
The complex field extractions (for example, one that extracts 6 fields at once) may be easier to maintain than multiple simple extractions (where you would have 6 different fields extracted by 6 different regexes).

Your own field extractions will probably be easier to maintain than those created by the Field Extraction Tool – just write your own regexes better than the FET.

Regexes can save you a lot of headaches compared to using non-regex field extractions (one user was trying to extract data using 200 non-regex evals compared to 6 regexes that accomplished the same thing!)
Tools
Regex101 Web Page
http://regex101.com
Regexr Web Page

http://regexr.com - Doesn’t do PCRE!!
Improve Regex Performance

https://www.loggly.com/blog/five-invaluable-techniques-to-improve-regex-performance/
Just for fun:
Try your regex prowess

- **Regex Golf**
  - [https://alf.nu/RegexGolf](https://alf.nu/RegexGolf)

- **Regex Crosswords**
  - [https://regexcrossword.com](https://regexcrossword.com)
  - [https://mariolurig.com/crossword/](https://mariolurig.com/crossword/)
Splunk Answers and Docs

Learn from others – ask questions – get answers


Splunk Documentation

▶ [https://docs.splunk.com/Documentation/Splunk/6.4.3/Knowledge/AboutSplunkregexpressions](https://docs.splunk.com/Documentation/Splunk/6.4.3/Knowledge/AboutSplunkregexpressions)

▶ Splunk **regex** Slack channel
Acknowledgements

- Lisa Guinn
  - Inspiration and guidance in preparing
  - Data set to use in examples
THE CHURCH OF
JESUS CHRIST
OF LATTER-DAY SAINTS
Questions?