

Regex  
Regexp  
Regexes  
Regular expression  
Reg(ular expressions? | ex(p | es)?

Gabriel Vasseur  
Sr Cyber Security Analyst, Thales

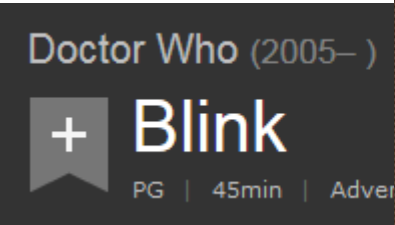
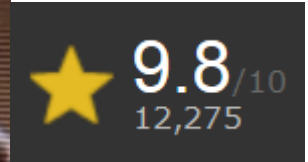
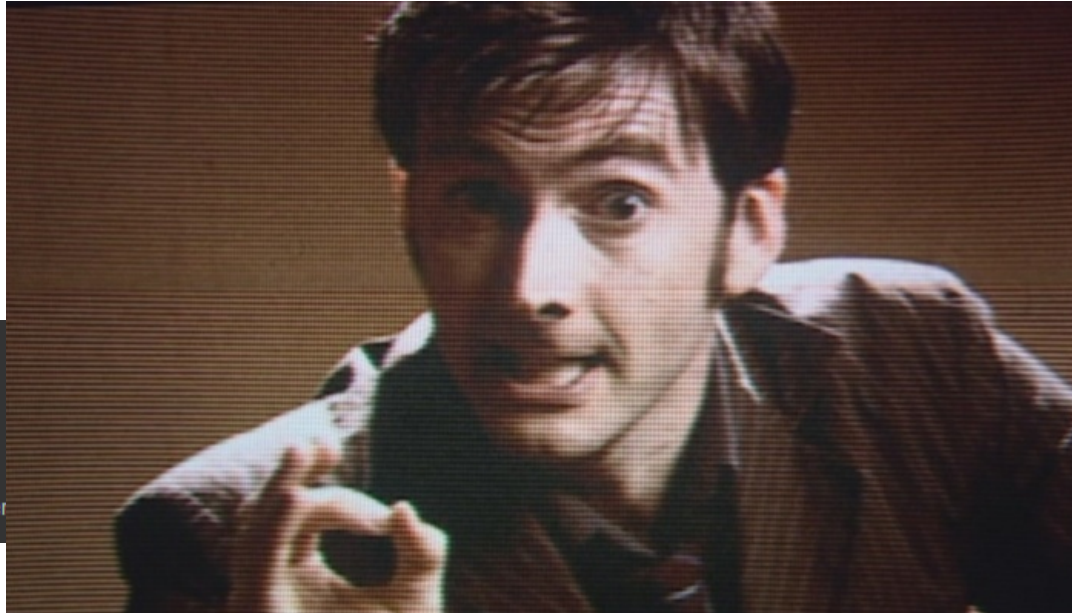


# Disclaimer

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 the 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.

# Surviving this talk

"DON'T BLINK!"



"BLINK AND YOU'RE DEAD".

# Surviving This Talk

## Hang on to the key messages:

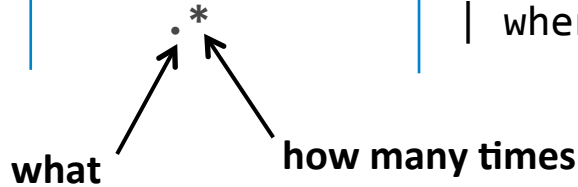
- Regexes are different from the simpler patterns we're used to
- It's always **WHAT**, optionally followed by **HOW MANY**
- Regexes are not anchored by default
- Wake up for the key tricks
- Legend: regex **match** **not-a-match** candidate-for-matching

Get the slides or the recording afterwards!

# Regex vs Rest of the (pattern) World



	any character	any amount of any character	Examples	special chars
bash dos splunk search	?	*	ls /opt/log/www?/*.log dir *.doc   search action="analy?e"	
SQL splunk "like"	_	%	WHERE phone LIKE '01256%'   where action LIKE "analy_e"   where like(action, "analy_e")	
Regexes	.	.	where match(action, "analy.e")	



# Regex vs Rest of the (pattern) World

	pattern vs value	Examples
bash dos splunk search	pattern matches <b>whole</b> value	search user="admin*" matches: <b>admin administrator</b> but not: <b>sysadmin</b>
SQL splunk "like"	pattern matches <b>whole</b> value	where like(action, "analy_e") matches: <b>analyze analyse</b> but not: <b>re-analyse</b>
Regexes	pattern matches <b>anywhere</b> in value...  ...unless anchored!	where match(user, "admin") matches: <b>admin administrator</b> <b>sysadmin</b>

# Regex vs Rest of the (pattern) World

Regex	Matches	Doesn't match
admin	admin administrator sysadmin	
^admin	admin administrator	sysadmin
admin\$	admin sysadmin	administrator
^adm?	admin	sysadmin administrator

Pointless

# Regex vs Rest of the (pattern) World

bash DOS splunk search	SQL splunk "like"	Regexes
admin	admin	<code>^admin\$</code>
<u>admin*</u>	admin%	<code>^admin</code>
*admin	%admin	<code>admin\$</code>
*admin*	%admin%	<code>admin</code>

*Pointless*

`^admin.*` `^admin.*$`  
`.*admin` `.*admin$`

*Over-complicated*

≠

**In conclusion, beware:**

admin\* matches admirator admin adminnnn badminton cadmium



# Special Characters

- Special Characters

- Line feed (unix EOL)            `\n`            (windows EOL: `\r\n`)
- Carriage return (old mac EOL)   `\r`
- Tab                                    `\t`

- Escaped Characters

- a literal dot                        `\.`
- a backslash                         `\\`
- etc... \_\_\_\_\_

# Character Classes

## Character classes

- (almost) any character `.`
- a digit `\d` `[0-9]` `[1234567890]`
- a word character `\w` `[A-Za-z0-9_]`
- white space `\s`
- a hex digit `[A-Fa-f0-9]`

## Negated character classes

- not a digit `\D` `[^0-9]`
- not a word character `\W` `[^A-Za-z0-9_]`
- not white space `\S`
- not a hex digit `[^A-Fa-f0-9]`

# Quantifiers

Pointless on their own!

- Zero or more \*
- One or more +
- Zero or one ?
- Exactly 4 times {4}
- Min. 3 times {3, }
- Min. 2, max. 5 times {2, 5}

# Anchors

- Match a position, zero-width!
- Start of line
- End of line
- Word boundary

^

↑  
Regexes are fun!

\$

↓  
Regexes are fun!

\b

↑ ↓ ↓ ↓  
Regexes are fun!

`admin\b` matches `admin` `sysadmin` `admin-sql`

but not `administrator` `admin_oracle`

# You Blinked!

`.*^$\[\]\+?{}  
-\^ in []`



# Putting it all Together (so far)


- UK/US spelling      `colou?r`      matches `color` `colour`
- An MD5 hash      `\b[A-Fa-f0-9]{32}\b`
- A base64 string      `[A-Za-z0-9/+]{0,3}`

# Grouping

## Alternatives

- Greeting in Canada: (hello|bonjour)
- [abc] same as (a|b|c)

## Quantifiable

- A sentence with at least one word:  $(\backslash w+ )^* \backslash w+ \backslash .$ 
- A Base64 paragraph :  $^([A-Za-z0-9/+] + \backslash r? \backslash n)^* [A-Za-z0-9/+] += \{0, 3\} \$$

# Grouping

## ■ Capturing with (...)

➤ Allows to refer to what was matched

➤ `Dear (Mrs?|Miss) (\w+)` applied to "Dear Mr Jones" → `\1="Mr" \2="Jones"`

## ■ Non-Capturing with (?:...)

➤ `Dear (?:Mrs?|Miss) (\w+)` applied to "Dear Mr Jones" → `\1="Jones"`

## ■ Extracting with (?P<...>...)

➤ `Dear (?P<title>Mrs?|Miss) (?P<name>\w+)`

➤ applied to "Dear Mr Jones" → `title="Mr" name="Jones"`



# Still here?



- Wake up! “Key tricks” are coming up!

# Key Tricks

• Quoted Phrases: ...DENIED "Social Networking" http://...

- Trick: everything is either the delimiter: " or not the delimiter: [^"]
- Answer: `"([^\"]*)"`
- what if "I have \" in the middle"? It gets complicated...

**Delimiter-separated list:** White List;Education;Technology/Internet

- naive: `^[^;]*;`
- what if at start or end? `(^|;)[^;]*($|;)`
- capturing the match `(?:^|;)([^\;]*)?:$|;`

**get the Nth field:** SYSTEM,User,Success Audit,ABC123456,Logon/Logoff

- get 4<sup>th</sup> field from start `^(?:[^\s,]*,){3}([^\s,]*)`
- get one-before-last field `,([^\s,]*),[^\s,]*$`

# To Infinity & Beyond!

- There is a lot more to regular expressions than this...
  - Mode modifiers (e.g. tweak case sensitivity)
  - Lookaround (e.g. match something but not if followed/preceded by something else)
  - The notion of greediness for quantifiers
  - backreferences (e.g. match repeated patterns)
  - ...
- Recommended resources:
  - <http://www.regular-expressions.info>
  - <http://regexr.com/>
  - <https://regex101.com/>

# How to Use Regular Expressions in Splunk!



splunk >

# Splunk > Filter Your Search Results

. \* ^ \$ \ [ ] + ? { } ( | )  
- ^ \ in [ ]

- ... | search some\_field=...
  - value case insensitive (field names are always case sensitive)
  - simple pattern, e.g. | search user=admin\*
  - simple comparisons, e.g. | search count>5
  - can't refer to other fields
- ... | where some\_field=...
  - value case sensitive
  - full eval syntax: can refer other fields, e.g. | where src!=dest
  - quotes: use " when referring to literal values and ' for field names
- ... | regex some\_field=...
  - straight forward filter based on a regular expression

# Splunk > eval match

- Part of the "eval" syntax, most useful in "if" or "case"
- DEMO MATCH
  - `index=main sourcetype=bluecoat* | dedup user | eval is_contractor=if(match(user, "^c"), "yes", "no") | table user is_admin is_contractor`
  - try naive regexes, notice issues and fix them (add `\d` and `a?`)
  - Add `| eval is_admin=if(match(user, "^a"), "yes", "no")`
  - Again, fix issues with `[uc]?\d`

# Splunk > eval replace

- Part of the "eval" syntax
- DEMO REPLACE
  - `index=main sourcetype=bluecoat DENIED Malicious | stats count by category`
  - `Replace stats: | eval cat=replace(category, "Malicious", "Delicious" ) | stats count by category cat`
  - `add (.*)` to the end of the regex and change replace string to `"\1 is delicious"`
  - Pointless but gives an idea of what is possible!

# Splunk > rex

- Extract new field(s) out of an existing field
- DEMO REX 1
  - `index=main sourcetype=bluecoat DENIED Malicious | stats count by cs_categories`
  - Notice the category issue
  - `| rex field=cs_categories "(?P<cat>Malicious)"`
  - Add `+`
  - Replace `.` with `[^;]`



# Splunk > rex

- DEMO REX 2: use Splunk's suggestions!
  - `index=main sourcetype=bluecoat | table _time _raw http_referrer`
  - open same in two tabs. In one of them "extract more fields"
  - If short of time, take one with "Social Networking" or "Entertainment", otherwise choose one with "Business/Economy" or "Technology/Internet"
  - **First problem:** not matching all the which.co.uk
  - Change first part of the regexp to either `^[^"]*"[^"]*"` or `^(?:\S+ ){12}"[^"]+`
  - **Second problem:** not matching - or other website
  - change after `://` to `\S+` → now see regular-expression.info also matching
  - change the whole URL to `\S+` → see the dash matching
  - go back to initial search and do `| rex field=_raw ""` ← copy regex and escape "
  - compare with `http_referrer`

# Splunk > rex

- Rex Conclusion
  - Know your data, know your regex: use Splunk's suggestion but tweak it!
  - Be as restrictive as you can where possible
  - Check many examples for edge cases
  - For permanent extraction you can copy-paste the regex and use it in an EXTRACT in a props.conf

# THANK YOU

- Remember the key messages:
  - The way regexes are different from the patterns you're used to
  - It's always **WHAT**, optionally followed by **HOW MANY**
  - Beware of unintended matches and edge cases
  - Regular expressions are not just for admins!
- Get the slides and/or recording
- Any questions? <https://answers.splunk.com/> with tag "regex" or "rex"

.conf2016