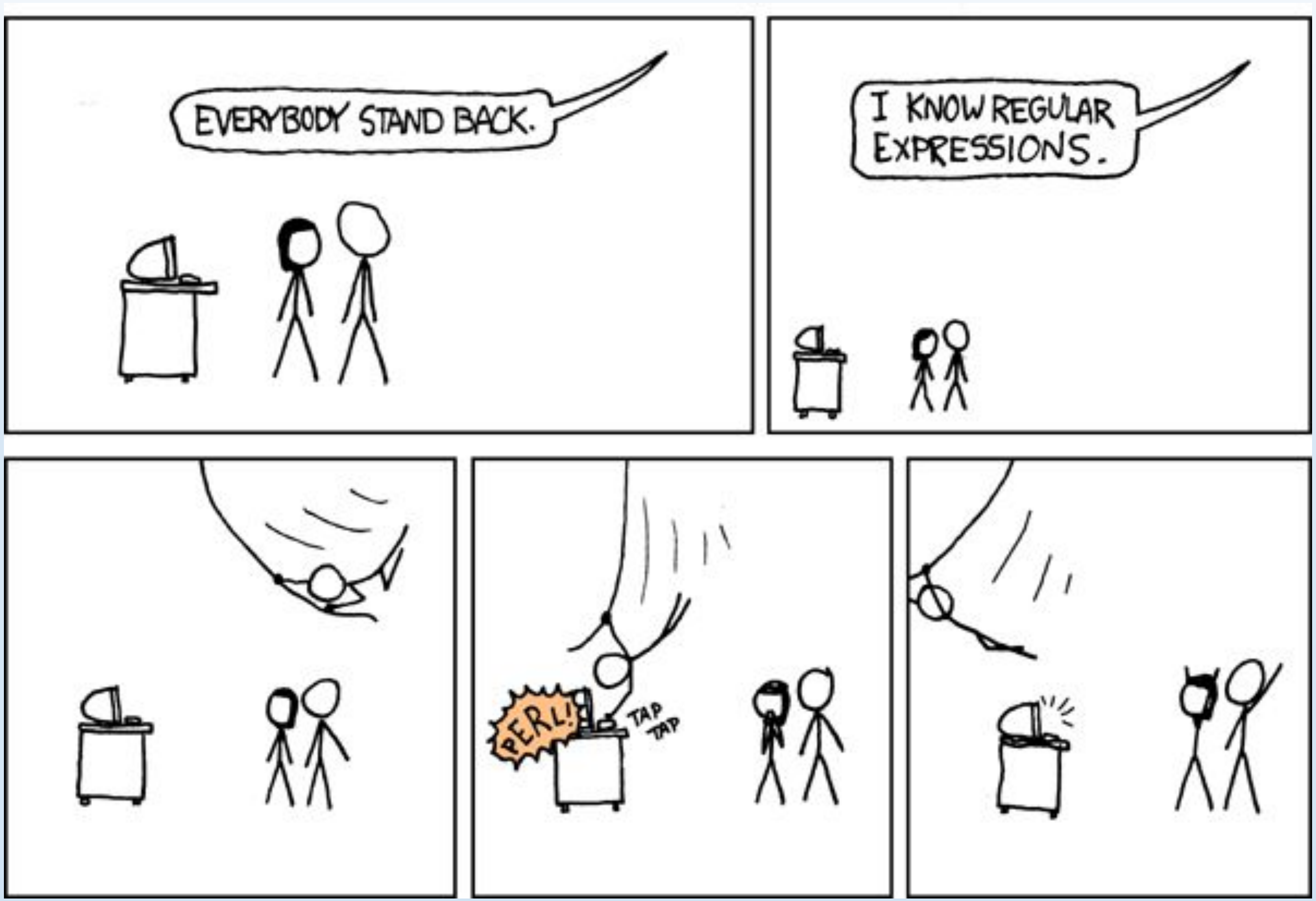
Regular Expression For Everyone

Sanjeev Kumar Jaiswal (Jassi)
Perl Programmer and Security Enthusiast

Agenda to make you superhero!

- Introduction to RegEx
- Match, substitute, transliterate
- Quantifiers, Character Class, Modifiers
- RegEx Examples in Perl
- Books, sites, blogs to learn RegEx


EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

Regex needs Intro?

- Stephen Col Kleene -> Regular Sets (1950)
- Ken Thompson -> ed (1971)
- Global Regular Expression Print (1973)
- Sed (1974) & awk (1977)(text processing tools)
- Search engines, word processors, text editors
- Henry Spencer -> Software library for Regular Expression (regex)
- Regular Expression (regex or regexp)

Where Regex is used!

- g/re/p (grep –P, egrep)
- sed and awk
- Perl and PCRE (Python, Ruby, PHP, JavaScript)
- Text Editor: Notepad++ ,vim,emacs,nano
- Search Engines
- Think of pattern, think of regex

Regex Fundamentals

Regex Operators

Match operator

- Represented by m at the beginning
- Matches Pan Card: m/^[a-z]{5}\d{4}[a-z]$/i

Substitute operator

- Represented by s/match_it/substitute_with/
- Remove Blank Line: s/^\s*$//

Transliteration Operator

- tr/// or earlier y///
- Change the case of string (uc function): tr/a-z/A-Z/

Metacharacters

Having special meaning?

Any character that has special meaning while using regex.

Ex: anchors, quantifiers, character classes.

The full list of metacharacters (12) is \, |, ^, $, *, +, ?, ., (,), [, {

Somewhere (in POSIX?), you will see 14 (including } and])

To use it with it’s literal meaning, you need to use backslash before using these characters.

. doesn’t mean fullstop in regex

* doesn’t mean multiplication in regex

Position Metacharacters (Anchors)

\ -> overrides the next metacharacter.

^ -> matches the position at the beginning of the input string.

. -> matches any single character except newline

$ -> matches the position at the end of the input string.

| -> specifies the or condition

() -> matches a pattern and creates a capture buffer for the match

[] -> Bracketed Character group/class

Character Classes

A character class is a way of denoting a set of characters in such a way that one character of the set is matched

- The dot
- The backslash sequence
  - Word Characters -> \w or [a-zA-Z0-9_]
  - Whitespace -> \s or [\t\n\f\r]
- Bracketed character classes, represented by [character_group]
  - [aeiou], [a-f-z], [a-z], [-f] -> Positive character class/group
  - [^\d] , [^^], [x^]-> negative character class/group
- POSIX Character classes, [:class:]
  - [[:alpha:]], [01[:alpha:]%], [:alnum:]
- Read more: http://perldoc.perl.org/perlrecharclass.html

Shorthand Character Classes

- \d for [0-9]
- \D for [^\d]
- \s for [\t\r\f\n]
- \S for [^\s]
- \w for [A-Za-z0-9_]
- \W for [^\w]

Zero –width Assertions

- \b Match a word boundary
- \B Match except at a word boundary
- \A Match only at beginning of string (think of ^)
- \Z Match only at end of string, or before newline at the end (think of $)
- \z Match only at end of string

Quantifies the pattern

Quantifiers

- . -> matches any character except new line
- * -> matches 0 or more
- ? -> matches 0 or 1
- + -> matches 1 or more
- {n} -> matches exactly n times
- {min,max} -> matches min to max times
- {,max} -> matches till max times
- {min,} -> matches min to no limit

Modifiers

Minimal Modifiers

- Changes the behavior of the regular expression operators.
  These modifiers appear at:
  1. the end of the match,
  2. substitution, and
  3. qr// operators
- i -> case insensitive pattern matching
- g -> globally match the pattern repeatedly in the string
- m -> treat string as multiple lines
- s -> treat string as single line
- x -> permitting whitespace and comments
- e -> evaluate the right-hand side as an expression

Want to know more: http://perldoc.perl.org/perlre.html#Modifiers

Regex Examples

Trim extra spaces from a string

Requirements:

- Trim if any space(s) found before the string
- Trim if any space(s) found after the string
- Ignore if space(s) exists in between the string.
- " c and cpp   " or "   c and cpp" or "   c and cpp   " should save as "c and cpp "

```
$word =~ s/^\s+//;
$word =~ s/\s+$//;
```

Or

```
$word =~ s/^\s+|\s+$//g;
```

Only Yes or No

Requirements:

You have to accept only if user types yes or no in any of the below manner:

- yes Yes YEs YES YeS yES yeS
- No no NO nO
- y Y
- n N

```
$answer = m/^(y(?:es)?|no?)$/i;
```

Simulate wc command

Requirements:

- We need to count
  - Number of words
  - Number of lines
  - Number of characters with space or without space.

```
open(IN, $filename) || die “Can’t open $filename: $!\n";
 foreach my $line (<IN>) {
    $chars_with_space += length($line);
    $space_count++ while $line =~ /\s+/g;
    $words++ while $line =~ /\S+/g;
    $lines+= $line=~/\n$/;
 }
 close(IN);
```

Nth Match in a string

Requirements:

- String is “121221132231222345981222209 8abcXYZ”
- Look for pattern12+ i.e. matches 12, 122, 1222 and so on
- Find 4th occurrence of the pattern

```
my $n = 4; # nth occurrence
my ($match) =~ /(?:.*?(12+)){$n}/;
print "#$n match is $NthMatch\n";
```

Or

```
my @matches = ($_ =~ /(12+)/g);
print "#$n match is $matches[3]\n";
```

Email Validation

Requirements:

Email could be anything but it should have some boundary like

- It will have @ and .
- Username can be a mixture of character, digit and _ (usually) of any length but it should start with only character i.e from a-z or A-Z (I restricted length from 8-15)
- Domain Name should be between 2 to 63
- After the last . there should not be @ and it could be in range of 2-4 only

```
$pattern =
/^([a-zA-Z][\w\_]{8,15})\@([a-zA-Z0-9.-]+)\.([a-zA-Z]{2,4})$/
;
```

* We can write another email pattern for newly available domains like .website, .shikha etc.

IP Address Validation

Requirements:

- Match IPv4 Address
- It should be between 0.0.0.0 and 255.255.255.255

Simplest One

```
$ip=/^(\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})$/;
```

Advanced One

```
$ip =~ /^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}
(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$/
```

Find WordPress Version

Requirements:

- Find a site which runs on WordPress
- Try to figure our WP version
  - Either by response header. Check for X-Meta-Generator keyword
  - Or by readme.html page
  - Or by wp-login.php page

```
($version) = $content =~ m/X-Meta-Generator:\s+(.*)/img;
($version) = $content =~ /version\s+(\d+\.\d+(?:\.\d+)?)/img;
($version) = $content =~
m#wp-(?:admin|content|includes)/(?!plugins|js).*?ver=(\d+\.\d+(
?:\.\d+)?(?:[-\w\.]+)?)#img;
```

Extract css/js files used in a site

Requirements:

- Request url and get content
- Parse through each js and css links
- Based on js and css files, guess the technology used

- Regex to find all possible CSS links

```
my (@css)= $content =~ m#<link\s+(?:.*?)href=['"]([^'"]*)#mgi;
foreach (@css){
   ($css) = $_ =~ m#(?:.*)/(.*)#g;
   ($css) = $_ =~ m#([^/]*\.css)#gi;
   ($css) = $css =~ m#(.*)\.css# if($css);
}
```

- Regex to find all possible JavaScript links

```
 my (@js)= $content=~ m#<script\s+(?:.*?)src=['"]([^'"]*)#mig;
```

Common credit card vendor RegEx

Amex Card: ^3[47][0-9]{13}$

Diners Club Card: ^3(?:0[0-5]|[68][0-9])[0-9]{11}$

Discover Card: 6(?:011|5[0-9]{2})[0-9]{12}

JCB Card: ^(?:2131|1800|35\d{3})\d{11}$

Maestro Card: ^(5018|5020|5038|6304|6759|6761|6763)[0-9]{8,15}$

Master Card: ^5[1-5][0-9]{14})$

Visa Card: ^4[0-9]{12}(?:[0-9]{3})?$

Snort Rule to alert detect Visa Card:

```
alert tcp any any <> any any
(pcre:"/4\d{3}(\s|-)?\d{4}(\s|-)?\d{4}(\s|-)?\d{4}/";msg:"VISA card number detected
in clear text"; content:"visa"; nocase; sid:9000000; rev:1;)
```

Preventing XSS

- Just an example, can add many such regex for better prevention
- Regex for simple CSS attack.

```
s/((\%3C)|<)((\%2F)|\/)*[a-z0-9\%]+((\%3E)|>)//gix;
```

- Blind regex for CSS attacks
  - `s/((\%3C)|<)[^<%3C]+((\%3E)|>?)//sig;` Regex for simple CSS attack.

```
s/((\%3C)|<)((\%2F)|\/)*[a-z0-9\%]+((\%3E)|>)//gix;
```

- Regex check for "<img src" CSS attack

```
s/((\%3C)|<)((\%69)|i|(\%49))((\%6D)|m|(\%4D))((\%67)|g|(\%47))
[^\n]+((\%3E)|>)//gi;
```

Find Duplicate words in a file

Requirements:

- Output lines that contain duplicate words
- Find doubled words that expand lines
- Ignore capitalization differences
- Ignore HTML tags

```
$/ = ".\n";
while (<>) {
   next if
  !s/\b([a-z]+)((?:\s|<[^>]+>)+)(\1\b)/\e[7m$1\e[m$2\e[7m$3\e[
  m/ig;
   s/^(?:[^\e]*\n)+//mg; # Remove any unmarked lines.
   s/^/$ARGV: /mg; # Ensure lines begin with filename.
   print;
}
```

More Regex

- # Famous one line Perl RegEx for Prime Number by Abigail

```
perl -le 'print "Prime Number"
  if (1 x shift) !~ /^1?$|^(11+?)\1+$/' [number]
```

- # decimal number

```
/^[+-]?\d+\.?\d*$/
```

- # Extract HTTP User-Agent

```
/^User-Agent: (.+)$/
```

- # Replace <b> with <strong>

```
$html =~ s|<(/)?b>|<$1strong>|g
```

Regex Tools

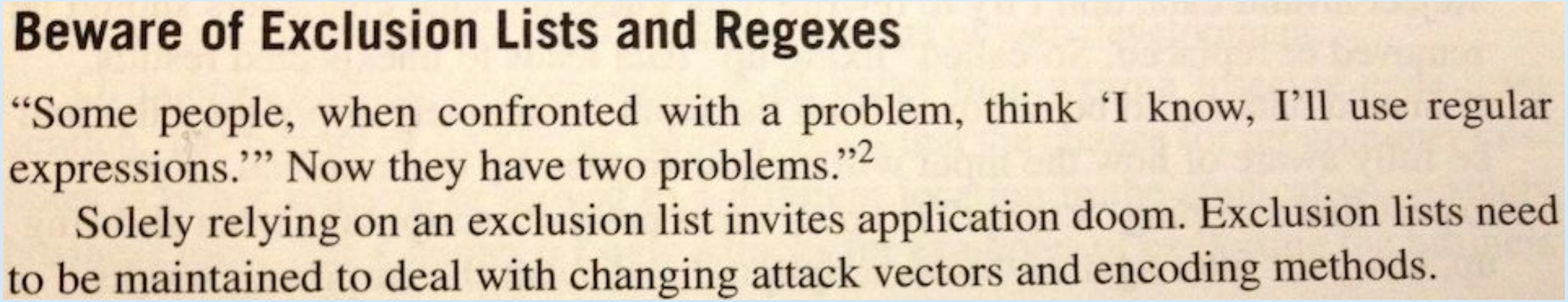
- My Favorite: http://regexr.com/
- Expresso for Windows: http://www.ultrapico.com/Expresso.htm
- Firefox Addon: https://addons.mozilla.org/en-US/firefox/addon/rext/
- Ruby based regex tool: http://www.rubular.com/
- Python based regex tool: http://www.pythonregex.com/
- JavaScript based regex tool: http://regexpal.com/

Final Note

- Don’t make simple things complex by using Regex
- Don’t try to do everything in one regex, better use multi regex
- Test Regex in regex tool before using
- Use Post processing whenever is possible
- Alternations? Think of grouping the regex
- Capture only it’s necessary
- Use /x for proper whitespace and comments, if possible
- Use regex wisely when dealing with lazy or greedy quantifiers
- Regex is not a natural language parser

*Next move is to learn backtrack and lookaround concepts in regex

Just a Note!



Beware of Exclusion Lists and Regexes

"Some people, when confronted with a problem, think 'I know, I'll use regular expressions.'" Now they have two problems."[2]

Solely relying on an exclusion list invites application doom. Exclusion lists need to be maintained to deal with changing attack vectors and encoding methods.

Resources

Books to read

1. Compilers Principles Techniques & Tools by Aho,Ullman
2. Mastering Regular Expression by Jeffrey Friedl
3. Regular Expression Recipes by Nathan A. Good
4. Mastering Python Regular Expression by [PACKT]
5. Learn Regex the hard way online
6. Wiki Book on Regular Expression

Resources/References

1. http://perldoc.perl.org/perlre.html
2. http://www.regular-expressions.info/
3. https://www.owasp.org/index.php/OWASP_Validation_Regex_Repository
4. http://rick.measham.id.au/paste/explain.pl
5. http://www.catonmat.net/series/perl-one-liners-explained
6. https://www.owasp.org/index.php/OWASP_Validation_Regex_Repository
7. http://rick.measham.id.au/paste/explain.pl
8. YouTube Video

Sharing is Caring!

1. FB: https://www.facebook.com/jassics
2. Twitter: https://www.twitter.com/jassics
3. Mailid: jassics [at] gmail [dot] com
4. Slideshare: https://www.slideshare.net/jassics
5. Linkedin: https://www.linkedin.com/in/jassics
6. Youtube: https://youtube.com/c/flexmind