

Regular Expression for URL Filtering and DNS Security

Regular Expressions

A regular expression is a pattern (a phrase, number, or more complex pattern) the CLI String Search feature matches against show or more command output. Regular expressions are case-sensitive and allow for complex matching requirements. Simple regular expressions include entries like Serial, misses, or 138. Complex regular expressions include entries like 00210... , (is), or [Oo]utput.

A regular expression can be a single-character pattern or a multiple-character pattern. That is, a regular expression can be a single character that matches the same single character in the command output or multiple characters that match the same multiple characters in the command output. The pattern in the command output is referred to as a string. This section describes creating both single-character patterns and multiple-character patterns. It also discusses creating more complex regular expressions using multipliers, alternation, anchoring, and parentheses.

Note In the local domain list used in Umbrella Integration, only simple domain/wildcard-style matching is supported, not full regular expression parsing. A hyphen is allowed as a normal DNS label character where valid, but regular expression constructs such as ^, $, [], ranges, quantifiers, alternation, grouping, and regex escaping are not supported.

Single-Character Patterns

The simplest regular expression is a single character that matches the same single character in the command output. You can use any letter (A-Z, a-z) or digit (0-9) as a single-character pattern. You can also use other keyboard characters (such as ! or ~) as single-character patterns, but certain keyboard characters have special meaning when used in regular expressions. The table below lists the keyboard characters that have special meaning.

Table 1: Characters with Special Meaning

Character	Special Meaning
.	Matches any single character, including white space.
*	Matches 0 or more sequences of the pattern.

Character	Special Meaning
+	Matches 1 or more sequences of the pattern.
?	Matches 0 or 1 occurrences of the pattern.
^	Matches the beginning of the string.
$	Matches the end of the string.
_(underscore)	Matches a comma (,), left brace ({), right brace (}), left parenthesis (( ), right parenthesis ( )), the beginning of the string, the end of the string, or a space.

To use these special characters as single-character patterns, remove the special meaning by preceding each character with a backslash (\). The following examples are single-character patterns matching a dollar sign, an underscore, and a plus sign, respectively.

\$ _ \+

You can specify a range of single-character patterns to match against command output. For example, you can create a regular expression that matches a string containing one of the following letters: a, e, i, o, or u. Only one of these characters must exist in the string for pattern matching to succeed. To specify a range of single-character patterns, enclose the single-character patterns in square brackets ([]). For example,

[aeiou]matches any one of the five vowels of the lowercase alphabet, while [abcdABCD] matches any one of the first four letters of the lower- or uppercase alphabet.

You can simplify ranges by entering only the endpoints of the range separated by a dash (-). Simplify the previous range as follows:

[a-dA-D]

To add a dash as a single-character pattern in your range, include another dash and precede it with a backslash:

[a-dA-D\-]

You can also include a right square bracket (]) as a single-character pattern in your range, as shown here:[a-dA-D\-\]]

The previous example matches any one of the first four letters of the lower- or uppercase alphabet, a dash, or a right square bracket.

You can reverse the matching of the range by including a caret (^) at the start of the range. The following example matches any letter except the ones listed:

[^a-dqsv]

The following example matches anything except a right square bracket (]) or the letter d:

[^\]d]

Multiple-Character Patterns

When creating regular expressions, you can also specify a pattern containing multiple characters. You create multiple-character regular expressions by joining letters, digits, or keyboard characters that do not have special meaning. For example, a4% is a multiple-character regular expression. Insert a backslash before the keyboard characters that have special meaning when you want to indicate that the character should be interpreted literally.

With multiple-character patterns, order is important. The regular expression a4% matches the character a followed by a 4 followed by a % sign. If the string does not have a4%, in that order, pattern matching fails. The multiple-character regular expression a.uses the special meaning of the period character to match the letter a followed by any single character. With this example, the strings ab, a!, or a2 are all valid matches for the regular expression.

You can remove the special meaning of the period character by inserting a backslash before it. For example, when the expression a\. is used in the command syntax, only the string a. will be matched.

You can create a multiple-character regular expression containing all letters, all digits, all keyboard characters, or a combination of letters, digits, and other keyboard characters. For example, telebit3107v32bis is a valid regular expression.

Multipliers

You can create more complex regular expressions that instruct Cisco IOS software to match multiple occurrences of a specified regular expression. To do so, you use some special characters with your single-character and multiple-character patterns. The table below lists the special characters that specify “multiples” of a regular expression.

Table 2: Special Characters Used as Multipliers

Character	Description
*	Matches 0 or more single-character or multiple-character patterns.
+	Matches 1 or more single-character or multiple-character patterns.
?	Matches 0 or 1 occurrences of a single-character or multiple-character pattern.

The following example matches any number of occurrences of the letter a, including none:

a*

The following pattern requires that at least one letter a be in the string to be matched:

a+

The following pattern matches the string bb or bab:

ba?b

The following string matches any number of asterisks (*):

******

To use multipliers with multiple-character patterns, you enclose the pattern in parentheses. In the following example, the pattern matches any number of the multiple-character string ab:

(ab)*

As a more complex example, the following pattern matches one or more instances of alphanumeric pairs, but not none (that is, an empty string is not a match):

([A-Za-z][0-9])+

The order for matches using multipliers (*, +, or ?) is to put the longest construct first. Nested constructs are matched from outside to inside. Concatenated constructs are matched beginning at the left side of the construct. Thus, the regular expression matches A9b3, but not 9Ab3 because the letters are specified before the numbers.

Alternation

Alternation allows you to specify alternative patterns to match against a string. You separate the alternative patterns with a vertical bar (|). Exactly one of the alternatives can match the string. For example, the regular expression codex|telebit matches the string codex or the string telebit, but not both codex and telebit.

Anchoring

You can instruct Cisco IOS software to match a regular expression pattern against the beginning or the end of the string. That is, you can specify that the beginning or end of a string contain a specific pattern. You "anchor" these regular expressions to a portion of the string using the special characters shown in the table below.

Table 3: Special Characters Used for Anchoring

Character	Description
^	Matches the beginning of the string.
$	Matches the end of the string.

For example, the regular expression ^conmatches any string that starts with con, and $sole matches any string that ends with sole.

In addition to indicating the beginning of a string, the ^ symbol can be used to indicate the logical function "not" when used in a bracketed range. For example, the expression **[^abcd]** indicates a range that matches any single letter, as long as it is not the letters a, b, c, or d.

Contrast these anchoring characters with the special character underscore (_). Underscore matches the beginning of a string (^), the end of a string ($), parentheses (( )), space ( ), braces ({}), comma (,), or underscore (_). With the underscore character, you can specify that a pattern exist anywhere in the string. For example,

_1300_ matches any string that has 1300 somewhere in the string. The string 1300 can be preceded by or end with a space, brace, comma, or underscore. So, although {1300_matches the regular expression **_1300_**, 21300 and 13000 do not.

Using the underscore character, you can replace long regular expression lists. For example, instead of specifying **^1300()()1300${1300,,1300,{1300},1300,(1300** you can specify simply **_1300_**.

Parentheses for Recall

As shown in the "Multipliers" section, you use parentheses with multiple-character regular expressions to multiply the occurrence of a pattern. You can also use parentheses around a single- or multiple-character pattern to instruct the Cisco IOS software to remember a pattern for use elsewhere in the regular expression.

To create a regular expression that recalls a previous pattern, you use parentheses to indicate memory of a specific pattern and a backslash (\) followed by a number to reuse the remembered pattern. The number specifies the occurrence of a parentheses in the regular expression pattern. If you have more than one remembered pattern in your regular expression, then \1 indicates the first remembered pattern, and \2 indicates the second remembered pattern, and so on.

The following regular expression uses parentheses for recall:

a(.)bc(.)\1\2

This regular expression matches an a followed by any character (call it character no. 1), followed by bc followed by any character (character number 2), followed by character no. 1 again, followed by character

number. 2 again. So, the regular expression can match aZbcTZT. The software remembers that character number 1 is Z and character number 2 is T and then uses Z and T again later in the regular expression.

• DNS Security Local-Domain Bypass Patterns, on page 5

DNS Security Local-Domain Bypass Patterns

Local-domain bypass is part of DNS Security/Umbrella/OpenDNS DNS redirection. It is evaluated only against the DNS query name, also called the DNS QNAME, present inside the DNS request.

It is not evaluated against full URLs, URI paths, HTTP headers, or browser-visible URLs.

Example:

```
User accesses: http://bad.example/path
DNS query seen: bad.example
```

Only bad.example is available for local-domain matching. http:// and /path are not part of DNS and cannot be matched by local-domain bypass.

Although the config uses parameter-map type regex, local-domain bypass does not support full regular-expression semantics. The regex parameter-map is used as a common config container. The local-domain backend consumes those entries as limited domain/wildcard-style strings.

Supported patterns for DNS security Local-Domain Bypass

The supported local-domain patterns are:

- cisco.com
- .*cisco.com
- .*.cisco.com

Multiple entries can be configured as a comma-separated list in the management/UI layer, or as separate patterns in the device-side regex parameter-map.

The types of patterns are:

- Base domain: Use this for the base domain.
- Suffix match: This is a suffix-style wildcard. It matches names ending in cisco.com.
- Subdomain: This matches subdomains ending in .cisco.com.

Pattern	Intended use	Matches	Does not match
`cisco.com`	Base domain	`cisco.com`	Do not rely on it for `www.cisco.com`
`.*cisco.com`	Suffix match	`cisco.com`, `www.cisco.com`, `mycisco.com`	Not label-boundary strict
`.*.cisco.com`	Subdomains under `cisco.com`	`www.cisco.com`, `a.b.cisco.com`	`cisco.com`, `mycisco.com`, `cisco.com.au`

To bypass both the base domain and its subdomains, configure both cisco.com and .*.cisco.com.

.

Unsupported patterns for DNS security Local-Domain Bypass

Do not configure full URLs:

```
http://bad.example/path
 https://bad.example
 bad.example/path
```

The full URLsdo not work for local-domain bypass because DNS does not carry protocol or path information.

Do not configure full regex constructs:

```
^yahoo\.com$
 [a-z].example
 .*(yahoo|google).com
 yahoo.com$
```

For local-domain bypass, ^, $, [], (), |, and similar regex operators are not treated as full regex syntax. They either become literal characters or do not match the intended DNS name.

Do not start a pattern with * or +:

```
*cisco.com
 +cisco.com
```

These patterns are rejected by the common regex parameter-map parser. Use the following instead:

```
.*cisco.com
 .*.cisco.com
```

Acceptable patterns for DNS Security Local-Domain Bypass

Escaped period

A backslash before a period is acceptable by the local-domain matcher:

```
yahoo\.com
```

Local-domain normalizes \. to . before matching. So the behaviour is:

```
yahoo.com
```

However, local-domain bypass prefers the simple form:

```
yahoo.com
```

Characters

Normal domain characters are acceptable:

```
example.com
 example-site.com
 site123.example.com
```

Hyphen is acceptable when used as part of a DNS label:

```
corp-site.example.com
```

Avoid URL-only characters and regex metacharacters unless they are actually part of the DNS name and tested for the specific use case.

Behavior when no pattern matches

If a DNS query does not match the local-domain bypass list, the packet follows with normal DNS Security behavior. For example, if DNS redirect to Cisco Umbrella is configured, the DNS request is redirected to Cisco Umbrella.

There is no separate alarm or notification when there is no pattern match for local-domain, because it is normal for any DNS query outside the bypass list.

Behavior when no pattern matches