BlackLynx for Splunk Usage Instructions V1.0

BlackLynx technology combines high performance heterogeneous computing (accelerated CPUs, GPUs, and FPGAs) with standard applications and protocols to achieve high performance analytics

Prerequisites
Install Splunk BlackLynx App version 1.0.2

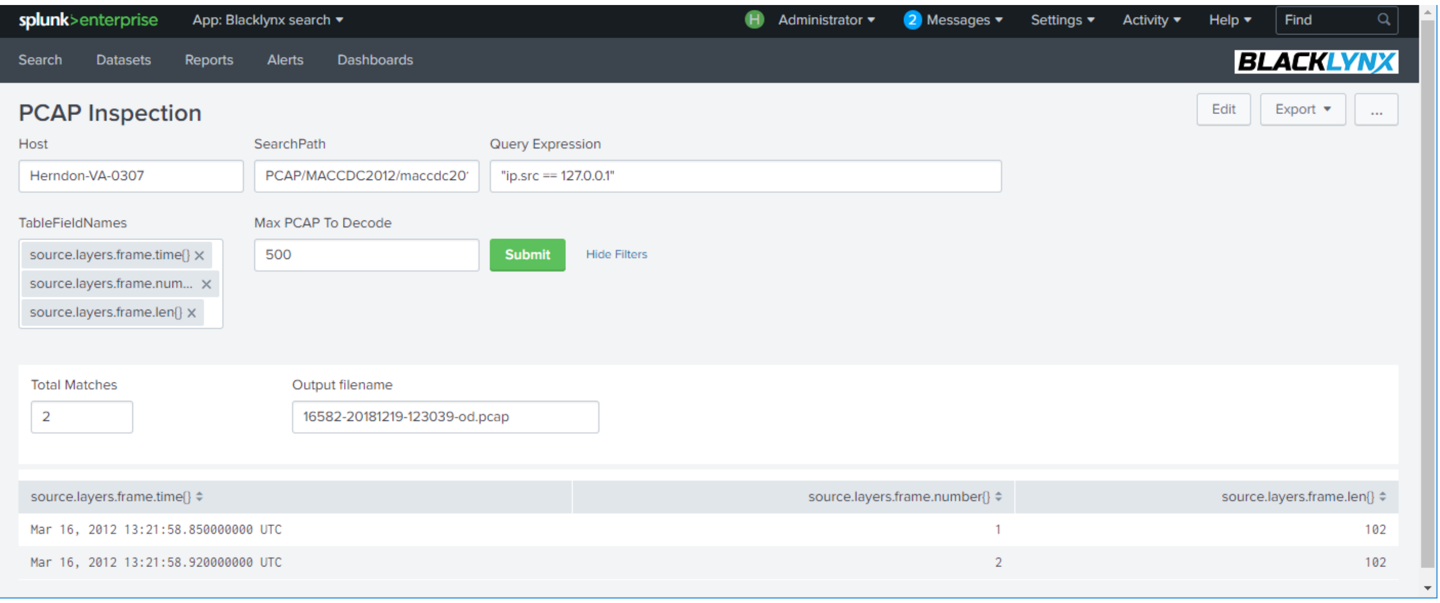
BlackLynx for Splunk brings the enhanced speed and search capabilities of the BlackLynx solution to the Splunk Search, Dashboard, and Alerts features. This is done using the BlackLynx Splunk Apps *blstructsearch and blunstructsearch*.

This document details the syntax of the BlackLynx Apps using the PCAP Inspection Dashboard as the primary example. It will cover:

- **The specifications for each input field,**
- **A Detailed specification of the "Query Expression" input to the blstructsearch App. The query specifies the search parameters used to thin and return just the desired data from an arbitrarily large file or set of files, and**
- **Examples of queries used in BlackLynx Splunk Dashboards.**

The PCAP Inspection Dashboard provides the ability to extract layer 2 though 7 fields from a PCAP file without prior decoding of the PCAP format. A capture of the input and results screens appears below:

PCAP Inspection Dashboard



PCAP Inspection

Host: Herndon-VA-0307
SearchPath: PCAP/MACCDC2012/maccdc20
Query Expression: "ip.src == 127.0.0.1"
TableFieldNames: source.layers.frame.time{} source.layers.frame.num... source.layers.frame.len{}
Max PCAP To Decode: 500

Total Matches: 2
Output filename: 16582-20181219-123039-od.pcap

source.layers.frame.time{}	source.layers.frame.number{}	source.layers.frame.len{}
Mar 16, 2012 13:21:58.850000000 UTC	1	102
Mar 16, 2012 13:21:58.920000000 UTC	2	102

BlackLynx for Splunk PCAP Inspection Dashboard

This section details the basic inputs and outputs for the BlackLynx for Splunk Dashboards

The PCAP Inspection dashboard allows the administrator to search and extract native PCAP records without having to go through a lengthy decoding process. The inputs and outputs from the dashboard will show the usage of the BlackLynx search apps.

BlackLynx PCAP Inspection Inputs

Input Field	Description
Host	BlackLynx server hostname or IP address
SearchPath	Pathname of PCAP file(s) to search
Query Expression	Search criteria expression
TableFieldNames	Specifies fields to show in the results table. Select fields from dropdown list. Remove from list by clicking 'X' next to field name. Defaults to ALL fields.
Max PCAP to Decode	Limits the number of search results to decode and return to Splunk.

BlackLynx PCAP Inspection Results Display

Results Field	Description of Field
Total Matches	The number of PCAP packets that matched the parameters in the Query String.
Output File	The name of the output PCAP file generated by the BlackLynx search. It will contain just the packets that matched the search criteria. This file is suitable for searching in a follow on query.
Splunk Results Table	Splunk created Table of the fields requested in the TableFieldNames dashboard input. The number of records in the table may be less than the total reported matches if the total matches exceeded the Max PCAP to Decode input.

BlackLynx for Splunk PCAP Inspection Dashboard - continued

BlackLynx PCAP Inspection Search Command

The BlackLynx Search App – **blstructsearch** – is a Splunk Generator and begins the search command and takes inputs host, filename, expr, and max_pcap_decode. It returns the total matches, the output file name (if produced), and all of the JSON decoded output records. The remainder of the commands use Splunk primitives to flatten the JSON and display the desired results. Here is the search command based inputs to this example:

```
| blstructsearch filename=PCAP/MACCDC2012/maccdc2012_00000.pcap
  host=Herndon-VA-0307 expr="ip.src == 127.0.0.1" max_pcap_decode=500
| spath | fields - _raw | fields - blstructsearch.total_matches,
blstructsearch.output_filename | table source.layers.frame.time{},
source.layers.frame.number{}, source.layers.frame.len{}
```

BlackLynx for Splunk App Command Syntax

BlackLynx for Splunk brings the enhanced speed and search capabilities of the BlackLynx solution to the Splunk Search, Dashboard, and Alerts features. This section details the syntax of the blstructsearch and blunstructsearch Apps and capability of the "Query Expression" input.

<u>BlackLynx App Search Command Syntax</u>

BlackLynx searches are classified as structured or unstructured based on the type of file and the way it is searched. Unstructured searches treat a file as a sequence of bytes without delimiters. Structured searches search within the records or fields of a record based on the type of file – CSV, JSON, XML or PCAP. All search primitives, specified in the <Query Expression> work on both types, but the syntax in the <Query Expression> differs slightly.

BlackLynx App Syntax for a structured search:

| blstructsearch host=<**Host>** filename=**<SearchPath>** expr=**<Query Expression>**
[max_pcap_decode=<**Max PCAP To Decode>] [filetype=<csv|json|xml|pcap>]**

BlackLynx App Syntax for an unstructured search:

| blunstructsearch host=<**Host>** filename=**<SearchPath>** expr=**<Query Expression>**

The '|' at the beginning of the line indicates that these apps are Splunk Generators and must be first in a string of commands.

BlackLynx for Splunk App Command Syntax - continued

BlackLynx App Search Command Parameters

Input Field	Description
Host	BlackLynx server hostname or IP address (required)
SearchPath	Pathname of PCAP file(s) to search (required) • **The path is specified relative to /ryftone on the BlackLynx server**. (ex "regression/Wikipedia-20150518.bin" refers to "/ryftone/regression/Wikipedia-20150518.bin" • For structured searches the name must end with a recognized extension like **.pcap**, **.xml**, **.csv**, or **.json.** Alternatively, the filetype parameter can be used to indicate the type of file. • Wildcard characters are allowed (e.g. PCAP/MACC*/maccdc2012*.pcap) • Multiple filenames can be specified by **enclosing with double quotes and delimiting with comma.**
Query Expression	Search criteria expression (required) Refer to query expression syntax and examples in this document.
max_pcap_decode	Limits the number of search results to decode and return. (optional) • Only used for PCAP searches • If you specify 0, only the **Total Matches** meeting the search criteria is displayed, **BUT NO output PCAP file is produced**. This is the fastest operation. This is recommended for trying various query expressions. • If you specify 1 or greater, the **Total Matches** meeting the search criteria is displayed, and an output PCAP file is produced on the BlackLynx server (**in the /ryftone directory**) containing all matching frames. (This file may be searched again to further limit results by making it the SearchPath in the subsequent command) • **The total decoded results returned to Splunk is limited to the Max PCAP To Decode value. (ex. -** If your query results in 250,000 matches, set Max PCAP To Decode to 100 to reduce decoding and Splunk formatting results time.) • Defaults to 500 decoded records
filetype	Sets the type of file for structured searches independent of the file's extension. (optional)

BlackLynx App Query Expression

The Query expression has 3 forms – **PCAP, Unstructured, and Structured**. This section starts with the syntax and use of each form, then goes into the details of each section of the query. The BlackLynx Open API Library User Guide has more depth and details on each part of the query.

PCAP Query Expression Syntax

The PCAP query syntax is used when analyzing PCAP files. BlackLynx can search based on frame (PCAP encoding additions to the stored packets), Layer 2 through Layer 4 parameters, and/or using any of the search primitives on the payload section of the packet. The output is a file containing the PCAP packets that matched the query and decoded fields in a JSON format.

<PCAP Fieldname> <Operator> <Value> [&& | || <PCAP Fieldname> <Operator> <Value>] … [&& | || (**RECORD.payload** <Relational Operator> <Search Primitive>(“<Payload Search Expression>”))]

Structured/Unstructured Query Expression Syntax

The Structured query syntax is used when analyzing files with a fixed format such as JSON, XML or CSV. When the format is known, BlackLynx can search based on either the entire record or fields within the record. If output data is requested, the file will contain the entire record where the match was detected.

Unstructured queries can be used on any files and force a straight binary search of the file. If output data is requested, the default will be to show the matched strings. Optional parameters allow for context on either side of the match.

(<input_specifier> <relational operator> <primitive>([expression[, options]]))

Multiple expressions may be grouped using AND/OR between expressions and parenthesis [()] for grouping logic.

PCAP Query Expressions

PCAP Fieldnames (Protocol Layer 2,3,4 field described below)

frame – a meta-layer associated with the frame headers attached to each packet in a PCAP file. Represented in the command as frame.<sub-field> in command. The associated field value can be one of:

- **time** – the frame time in UTC, with format: Mmm DD, YYYY HH:MM:SS[.s[s[s[s[s[s[s[s[s]]]]]]]]] where Mmm is the three-letter capitalized month, e.g., Jan 2, 2017 15:59:03.000
- **time_delta** – the delta time from the previous packet in the input PCAP file associated with the packet, as a time offset in seconds with up to nine positions after the decimal point
- **time_delta_displayed** – the delta time from the previous packet matched in the query filter associated with the operation, as a time offset in seconds with up to nine positions after the decimal point
- **time_epoch** – the frame time as the number of seconds elapsed since midnight January 1, 1970
- **time_invalid** – 1-bit: 1 if the frame time is invalid (out of range), 0 if valid
- **time_relative** – the delta time from the first packet in the input PCAP file associated with the packet, as a time offset in seconds with up to nine positions after the decimal point

eth - Ethernet layer 2 protocol. The associated field value can be one of:

- **src** - source address (value: 6 2-hex-digit octets separated by colons)
- **dest** [synonym: **dst**] - destination address
- **addr** - either the source or the destination address
- **type** - 16 bit Ethertype (integer or hex if leading 0x used)

vlan - Virtual LAN, as defined by IEEE 802.1Q, which resides at or alongside layer 2. The associated field value can be one of:

- **etype** - VLAN Ethernet subtype (integer or hex if leading 0x used)
- **tci** - 16 bit value representing all tag control information (TCI)
- **priority** | **pcp** - priority code point 3-bit (0-7) field
- **dei** | **cfi** - drop eligible indicator (DEI) one bit (0-1) field
- **id** | **vid** - VLAN identifier, lower 12 bits (0-4095) of the TCI. Note: if double tagging is used, both the inner and outer tags will be searched

mpls - Multi-protocol label switching, which is a hybrid layer 2/3 protocol. The associated field value can be one of:

- **label** - 20-bit value (0-1,048575) for the MPLS label
- **exp** | **tc** - 3-bit value (0-7) for the traffic class (and QoS and ECN)
- **bottom** | **s** - 1-bit: 1 when the current label is the last in the stack
- **ttl** - 8-bit (0-255) time-to-live indicator

PCAP Query Expressions - PCAP Fieldnames (continued)

ip - IPv4 layer 3 protocol. The associated field value can be one of:
- **src** - source address (value: [0-255].[0-255].[0-255].[0-255])
- **dest** [synonym: **dst**] - destination address
- **addr** - either the source or the destination address
- **proto** – protocol used in the data portion of the IPv4 datagram (value: [0-255])

ipv6 - IPv6 layer 3 protocol. The associated field value can be one of:
- **src** - source address (value: 8 2-hex-digit octets separated by colons)
- **dest** [synonym: **dst**] - destination address
- **addr** - either the source or the destination address
- **nxt** – protocol used in the data portion of the IPv6 datagram (value: [0-255])

icmp - ICMP protocol, for IPv4. The associated field value can be one of:
- **type** - the ICMP type (value: [0-255])
- **code** - the ICMP subtype (value: [0-255])

icmpv6 - ICMP protocol, for IPv6. The associated field value can be one of:
- **type** - the ICMP type
- **code** - the ICMP subtype

udp - UDP layer 4 protocol. The associated field value can be one of:
- **srcport** - the source port (value: [0-65535])
- **dstport** - the destination port
- **port** - either the source or the destination port

tcp - TCP layer 4 protocol. The associated field value can be one of:
- **srcport** - the source port (value: [0-65535])
- **dstport** - the destination port
- **port** - either the source or the destination port
- **flags** - Boolean quantities (0 or 1) representing the TCP flags construct, with the following supported subfields: **fin** - the FIN bit, s**yn** - the SYN bit, and **reset** - the RST bit

Operator Values

Equality: ==,	Inequality: !=, ne	Not: !, not
Greater than: >, gt	Less than: <, lt	
Greater or equal to: >=, ge	Less than or equal to: <=, le	

Note: “Not” operations are currently supported for use directly alongside Boolean types only like tcp.flags.[fin|syn|reset].

PCAP Query Expression (continued)

This section describes the query expression syntax for payload data in PCAP files.

RECORD.payload <Payload Relational Operator> <Payload Search Expression>
Search layer 4 payload using BlackLynx search primitives.

Payload Relational Operator

CONTAINS	The payload must contain Payload Search Expression. It may contain additional leading or trailing data
NOT_CONTAINS	The payload must not contain Payload Search Expression.
EQUALS	The payload must match Payload Search Expression either exactly for an exact search or within the specified distance for a fuzzy search, with no additional leading or trailing data.
NOT_EQUALS	The payload must be anything other than Payload Search Expression

Payload Search Expression

- **EXACT(expression[,options])**
  exact match of the expression. Specify option CASE="false" for case insensitive
- **PCRE2(expression)**
  match specified PCRE2 regular expression.
- **HAMMING(expression,DISTANCE="<number>")**
  match the expression "close enough" using Hamming search algorithm. DISTANCE specifies the maximum number of character substitutions allowed in order to match
- **EDIT_DISTANCE(expression,DISTANCE="<number>")**
  Match the expression by allowing a number of insertions, deletions and replacements
- **TIME(expression)**
  Match a time or time range

  Time format: HH:MM:SS or HH:MM:SS:ss (ss indicates hundredths of a second)
  Match a single time expression: TimeFormat *operator* ValueB (*operator* can be: =, !=, >=, >, <=, <)
  Example: (RECORD.payload CONTAINS TIME(HH:MM:SS > 09:15:00))
  Match a time range: ValueA *operator* TimeFormat *operator* ValueB (*operator can be: <, <=)*
  Example: (RECORD.payload CONTAINS TIME(11:15:00 < HH:MM:SS < 13:15:00))

PCAP Query Expression (continued)

This section describes the query expression syntax for PCAP files.

- **DATE(expression)**
  Match a date or date range

  Date format:

YYYY/[M]M/[D]D	YY/[M]M/[D]D
[D]D/[M]M/YYYY	[D]D/[M]M/YY
[M]M/[D]D/YYYY	[M]M/[D]D/YY

  Note: "/" can be replaced by any other single character like "-" or "_" (e.g. YYYY-MM-DD)

  Match a single date expression: DateFormat *operator* ValueB (*operator* can be: =, !=, >=, >, <=, <)
  Example: (RECORD.payload CONTAINS DATE(MM/DD/YY > 02/28/12))

  Match a date range: ValueA *operator* DateFormat *operator* ValueB (*operator can be: <, <=)*
  Example: (RECORD.payload CONTAINS DATE(02-28-1998 < MM-DD-YY < 09-19-2017))

- **NUMBER(expression[,options])**
  Match a number or range of numbers
  The SEPARATOR option defines the separating character (default is comma)
  The DECIMAL option defines the decimal point character (default is dot)
  If used, the DECIMAL and SEPARATOR character must be different.

  Match a single number expression: NUM *operator* "ValueA" (*operator* can be: =, !=, >=, >, <=, <)
  Example: (RECORD.payload CONTAINS NUMBER(NUM > "1058"))

  Match a range of numbers: "ValueA" *operator* NUM *operator* "ValueB" (*operator can be: <, <=)*
  (RECORD.payload CONTAINS NUMBER("1025" < NUM < "1050",SEPARATOR=",",DECIMAL="."))

- **CURRENCY(expression[,options])**
  Match a currency or currency range
  The SYMBOL option defines the monetary symbol (default is $)
  The SEPARATOR option defines the separating character (default is comma)
  The DECIMAL option defines the decimal character (default is dot)
  If used, the DECIMAL and SEPARATOR character must be different.

  Match a single currency expression: CUR *operator* "ValueA" (*operator* can be: =, !=, >=, >, <=, <)
  Example: (RECORD.payload CONTAINS CURRENCY(CUR > "1,058.54"))

  Match a currency range: "ValueA" *operator* CUR *operator* " (*operator can be: <, <=)*
  (RECORD.payload CONTAINS CURRENCY("$1025" < CUR < "$1058.54"))

PCAP Query Examples

These examples are taken from the BlackLynx for Splunk Dashboards

Example 1 – Search for DNS query responses (source port == 53) in a PCAP file

```
| blstructsearch filename=PCAP/wired-lan-session.pcap host=Herndon-VA-0313
    expr="udp.srcport == 53"
```

Find all frames in the file /ryftone/PCAP/wired-lan-session.pcap on hostname blsvr1 where the UDP source port equals 53. Since max_pcap_decode is not specified, a maximum 500 results would be decoded by default. "| spath" stores the information from the JSON formatted results into fields to be displayed.

Example 2 – Search for packets where a Social Security Number is transmitted in Clear Text

```
| blstructsearch filename=PCAP/newData/*.pcap host=blsvr1 expr="tcp.port != 443
   and (RECORD.payload CONTAINS PCRE2(\"[^-0-9]*\d{3}-\d{2}-\d{4}[^-0-9]*\"))"
```

Find all frames in all the files ending with .pcap in the /ryftone/PCAP/newData directory where the TCP source or destination port is not 443, and within the payload is a <3 digit-2 digit-4 digit> sequence (i.e. social security number format). The payload is searched using a PCRE2 regular expression match. Note the double quotes inside the PCRE2 expression are preceded by \ because the complete query expression must be enclosed by double quotes.

Example 3 – Search for any HTTP GET command sent to a particular IP address

```
 expr="ip.dst == 10.17.30.2 and (RECORD.payload CONTAINS EXACT(\"GET\"))"
```

This query expression finds matches for IP destination address 10.17.30.2 AND within the payload is the word "GET" (all CAPS since case sensitive is the default).

Example 4 – Find any Wake-On-LAN commands

```
| blstructsearch filename=wol*.pcap host=blsvr1 expr="(udp.port == 7 &&
    (RECORD.payload CONTAINS PCRE2(\"^\xff\xff\xff\xff\xff\xff\")))||
    (eth.dest == 'ff:ff:ff:ff:ff:ff' && ip.dest == '255.255.255.255' &&
    (RECORD.payload CONTAINS PCRE2(\"^\xff\xff\xff\xff\xff\xff\")))"
```

Find all frames in filenames beginning with "wol" and ending in ".pcap" in /ryftone which meet the definition of the Wake-on-LAN specification.

Example 5 – Find the number of packets not sent on localhost.

```
| blstructsearch filename=PCAP/MACCDC2012/maccdc2012_00000.pcap
    host=Herndon-VA-0307 expr="ip.src != 127.0.0.1" max_pcap_decode=0
```

Since **max_pcap_decode is 0**, only 1 result is returned, the field, blstructsearch.total_matches, indicating the number of frames in the PCAP file whose IP source address is not 127.0.0.1.

Syntax for BlackLynx Structured/Unstructured Search

The Syntax for **Structured and Unstructured file searching** is similar to the queries used for the PCAP payload data**. Structured files**, like JSON, XML, or CSV, can be searched for data anywhere within a record or within a field in each record. **Unstructured files** are searched as binary or text files and are considered a long string of bytes. A structured file may be searched as unstructured, but not vice versa.

The syntax for these queries is:

```
(<input_specifier> <relational operator> <primitive>([expression[, options]]))

Multiple expressions may be grouped using AND/OR between expressions and parenthesis
[()] for grouping logic.
```

input_specifier specifies how the input data is arranged. The possible values are:

input_specifier Values	Description
RAW_TEXT	The input is a sequence of raw bytes with no implicit formatting or grouping. **This must be used for unstructured searches**.
RECORD	The input is a series of records. Operate on all fields in each record.
RECORD.foo.bar	The input is a series of records. Operate only on the field whose hierarchy is spelled out by (in this example) foo.bar in each record. Note: for JSON and XML input records, field name hierarchies can be specified with '.' separators between them to specify a field hierarchy. For JSON, '[]' separators can be included as appropriate to specify array hierarchy. Whitespace is permitted in JSON fields if the individual field(s) containing whitespace are surrounded in double quotes. For CSV, <field_name> can be either a column name surrounded in double quotes, or a numeric index, where the leftmost column is column 1.

Syntax for BlackLynx Structured/Unstructured Search (continued)

relational_operator specifies how the input relates to the expression. The possible values are:

relational_operator Values	Description
EQUALS	The input must match expression either exactly for an exact search or within the specified distance for a fuzzy search, with no additional leading or trailing data. Note that this operator has meaning only for record- and field-based searches. If used with raw text input, an error will be generated. When searching raw text data, CONTAINS should be used instead of EQUALS.
NOT_EQUALS	The input must be anything other than expression. Note that this operator has meaning only for record- and field-based searches. If used with raw text input, an error will be generated.
CONTAINS	The input must contain expression. It may also contain additional leading or trailing data.
NOT_CONTAINS	The input must not contain expression. Note that this operator has meaning only for record- and field-based searches. If used with raw text input, an error will be generated. If you desire to determine whether an expression does not exist in raw text input, you should use CONTAINS, and check for zero results.

Syntax for BlackLynx Structured/Unstructured Search (continued)

primitive specifies the primitive associated with the clause. The primitives are the same as discussed for the PCAP Payload search   The possible values are:

primitive Values	Short Description
EXACT	Search for an exact match.
HAMMING	Perform a fuzzy search using the Hamming distance algorithm.
EDIT_DISTANCE	Perform a fuzzy search using the edit distance (Levenshtein) algorithm.
PCRE2	Search using a PCRE2-compliant regular expression.
DATE	Search for a date or a range of dates.
TIME	Search for a time or a range of times.
NUMBER	Search for a number or a range of numbers.
CURRENCY	Search for a monetary value or a range of monetary values.
IPV4	Search for an IPv4 address or a range of IPv4 addresses.
IPV6	Search for an IPv6 address or a range of IPv6 addresses.

Examples of BlackLynx Structured/Unstructured Search

These examples are taken from the BlackLynx for Splunk Dashboards

Example 1 – Structured search of a CSV file looking for exact value or fuzzy value for records between two dates. (File contains data on tweets determined to be Russian trolling)

```
| blstructsearch filename=twitter/russian-troll/*.csv host=Herndon-VA-0307
expr="((RECORD CONTAINS EXACT(\"hrc\",CASE=\"false\")) OR (RECORD CONTAINS
HAMMING(\"hillary\", DISTANCE=\"1\"))) AND (RECORD CONTAINS HAMMING(\"lock her
up\", DISTANCE=\"2\",CASE=\"false\")) AND (RECORD.\"publish_date\" CONTAINS
DATE(01/01/2017 <= MM/DD/YYYY <= 12/31/2017)) AND (RECORD.15 CONTAINS
EXACT(\"RightTroll\"))"
| eval monthNum=strftime(strptime(publish_date,"%m/%d/%Y %H:%M"),"%-m")
| eval monthName=strftime(strptime(publish_date,"%m/%d/%Y %H:%M"),"%-b")
| eval month=monthNum."-".monthName
| chart count by month
| rex field=month "(?<monthNum>\d+)-(?<month>.*)“
| fields - monthNum
```

Find all lines in the CSV files “/ryftone/twitter/Russian-troll/*.csv” where the ((record contains case insensitive exact match for “hrc” or fuzzy hamming match with a distance of 1 on “Hillary”) and a fuzzy match distance 2 on “lock her up” where the tweets were published in 2017 and were determined to be trolling the right wing voters.

Example 2 – Structured search of a CSV file where the search is limited to text in certain fields

```
| blstructsearch filename=twitter/russian-troll/*.csv host=Herndon-VA-0307
expr="(RECORD.\"content\" CONTAINS EDIT_DISTANCE(\"Donald Trump\",DISTANCE=\"2\"))
AND (RECORD.\"content\" CONTAINS EDIT_DISTANCE(\"Vladimir Putin\",DISTANCE=\"1\"))
AND (RECORD.15 CONTAINS EXACT(\"LeftTroll\")) AND (RECORD.\"publish_date\" CONTAINS
DATE(01/01/2016 <= MM/DD/YYYY <= 06/30/2018))“
| chart max(followers) AS followers by author
```

Find all lines in the CSV files “/ryftone/twitter/Russian-troll/*.csv” where the field “content” contains a fuzzy edit match with a distance of 2 for “Donald Trump” and a fuzzy edit match with a distance of 1 on “Vladimir Putin”) and the tweets were published in the first half of 2016 and were determined to be trolling the left wing voters. Note the text fuzzy edit distance searches were in the field “RECORD.CONTENT” which was named in the header line of the CSV files and that the RECORD.15 search was on the fifteenth field in each record. The named fields are only available when the first line of the CSV file contains column names for the subsequent records.

Examples of BlackLynx Structured/Unstructured Search (continued)

Example 3 – Unstructured regular expression search of a log file

```
|blunstructsearch host=R01-0001 filename=splunk_40*/* expr="(RAW_TEXT CONTAINS
PCRE2(\"client=(?!127\.\d+\.\d+\.\d+)(\d+\.\d+\.\d+\.\d+)latency\",LINE=\"TRUE\"))"
```

Find all lines in the log files "/ryftone/splunk_40*/*" where a text match for the regular expression

"client=(?!127\.\d+\.\d+\.\d+)(\d+\.\d+\.\d+\.\d+) latency" is found".

This search uses the blunstructsearch App to search a line in a log file for a line where the client tag is equal to an IP address that does not begin with 127. The option LINE="TRUE" indicates the results should include the entire line in the log file.