Querying PCAP Data Using Fixed Filter

Date of Publish: 2018-12-21

https://docs.hortonworks.com/

Contents

Using PCAP

The pcap data source can rapidly ingest raw data directly into HDFS from Kafka. As a result, you can store all of the raw packet capture data in HDFS and review or query it at a later date.

The pcap data is not displayed in the Metron dashboard, but you can query, view, or retrieve the data in order to port it to another application like Wireshark.

Capturing pcap Data

In your production environment there is likely to be one or more hosts configured with one or more span ports that receives raw packet data from a packet aggregation device. You can use one of HCP's packet capture programs to capture the pcap data; pycapa and DPDK. These programs are responsible for capturing the raw packet data off the wire and sending that data to Kafka where it can be ingested by HCP.

The following example uses Pycapa.

```
service pycapa start
```

If everything worked correctly, the raw packet data can be consumed from a Kafka topic called pcap. The data is binary.

```
$ /usr/hdp/current/kafka-broker/bin/kafka-console-consumer.sh -z
 zookeeper1:2181 --topic pcap
E)###>K######P#"ssLQlJ
                      P##0
E(  #@##x####>K###"PQlJ
                           ssLPF#
```

Processing pcap Data

After you capture some pcap data, the next step is to have HCP process the pcap data and store it in HDFS. Start the PCAP topology to begin this process. A Storm topology called 'pcap' is launched that consumes the raw pcap data from the Kafka topic and writes this data into sequence files in HDFS.

```
$ $METRON_HOME/bin/start_pcap_topology.sh
Running: /usr/jdk64/jdk1.8.0_77/bin/java -server -Ddaemon.name= -
Dstorm.options= -Dstorm.home=/usr/hdp/2.5.0.0-1245/storm -Dstorm.log.dir=/
var/log/storm -Djava.library.path=/usr/local/lib:/opt/local/lib:/usr/lib -
Dstorm.conf.file= -cp /usr/hdp/2.5.0.0-1245/storm/lib/log4j-core-2.1.jar:/
usr/hdp/2.5.0.0-1245/storm/lib/storm-core-1.0.1.2.5.0.0-1245.jar:/usr/
hdp/2.5.0.0-1245/storm/lib/minlog-1.3.0.jar:/usr/hdp/2.5.0.0-1245/storm/
lib/objenesis-2.1.jar:/usr/hdp/2.5.0.0-1245/storm/lib/ring-cors-0.1.5.jar:/
usr/hdp/2.5.0.0-1245/storm/lib/storm-rename-hack-1.0.1.2.5.0.0-1245.jar:/
usr/hdp/2.5.0.0-1245/storm/lib/disruptor-3.3.2.jar:/usr/hdp/2.5.0.0-1245/
storm/lib/kryo-3.0.3.jar:/usr/hdp/2.5.0.0-1245/storm/lib/log4j-over-
slf4j-1.6.6.jar:/usr/hdp/2.5.0.0-1245/storm/lib/reflectasm-1.10.1.jar:/
usr/hdp/2.5.0.0-1245/storm/lib/log4j-slf4j-impl-2.1.jar:/usr/
hdp/2.5.0.0-1245/storm/lib/log4j-api-2.1.jar:/usr/hdp/2.5.0.0-1245/storm/
lib/clojure-1.7.0.jar:/usr/hdp/2.5.0.0-1245/storm/lib/zookeeper.jar:/
usr/hdp/2.5.0.0-1245/storm/lib/servlet-api-2.5.jar:/usr/hdp/2.5.0.0-1245/
storm/lib/slf4j-api-1.7.7.jar:/usr/hdp/2.5.0.0-1245/storm/lib/
asm-5.0.3.jar org.apache.storm.daemon.ClientJarTransformerRunner
```

```
 org.apache.storm.hack.StormShadeTransformer /usr/metron/0.3.0/lib/metron-
pcap-backend-0.3.0.jar /tmp/d5f844e8b1a611e6a6d10a0a570e5f4d.jar
Running: /usr/jdk64/jdk1.8.0_77/bin/java -client -Ddaemon.name= -
Dstorm.options= -Dstorm.home=/usr/hdp/2.5.0.0-1245/storm -Dstorm.log.dir=/
var/log/storm -Djava.library.path=/usr/local/lib:/opt/local/lib:/usr/
lib:/usr/hdp/current/storm-client/lib -Dstorm.conf.file= -cp /usr/
hdp/2.5.0.0-1245/storm/lib/log4j-core-2.1.jar:/usr/hdp/2.5.0.0-1245/storm/
lib/storm-core-1.0.1.2.5.0.0-1245.jar:/usr/hdp/2.5.0.0-1245/storm/lib/
minlog-1.3.0.jar:/usr/hdp/2.5.0.0-1245/storm/lib/objenesis-2.1.jar:/usr/
hdp/2.5.0.0-1245/storm/lib/ring-cors-0.1.5.jar:/usr/hdp/2.5.0.0-1245/storm/
lib/storm-rename-hack-1.0.1.2.5.0.0-1245.jar:/usr/hdp/2.5.0.0-1245/storm/
lib/disruptor-3.3.2.jar:/usr/hdp/2.5.0.0-1245/storm/lib/kryo-3.0.3.jar:/usr/
hdp/2.5.0.0-1245/storm/lib/log4j-over-slf4j-1.6.6.jar:/usr/hdp/2.5.0.0-1245/
storm/lib/reflectasm-1.10.1.jar:/usr/hdp/2.5.0.0-1245/storm/lib/log4j-
slf4j-impl-2.1.jar:/usr/hdp/2.5.0.0-1245/storm/lib/log4j-api-2.1.jar:/usr/
hdp/2.5.0.0-1245/storm/lib/clojure-1.7.0.jar:/usr/hdp/2.5.0.0-1245/storm/
lib/zookeeper.jar:/usr/hdp/2.5.0.0-1245/storm/lib/servlet-api-2.5.jar:/
usr/hdp/2.5.0.0-1245/storm/lib/slf4j-api-1.7.7.jar:/usr/hdp/2.5.0.0-1245/
storm/lib/asm-5.0.3.jar:/tmp/d5f844e8b1a611e6a6d10a0a570e5f4d.jar:/usr/hdp/
current/storm-supervisor/conf:/usr/hdp/2.5.0.0-1245/storm/bin -Dstorm.jar=/
tmp/d5f844e8b1a611e6a6d10a0a570e5f4d.jar org.apache.storm.flux.Flux --
remote /usr/metron/0.3.0/flux/pcap/remote.yaml --filter /usr/metron/0.3.0/
config/pcap.properties
############     ###   ######  ###
############     ###   ############
######  ###      ###   ### ######
######  ###      ###   ### ######
###     ###################### ###
###     ######## ####### ###  ###
+-         Apache Storm        -+
+-  data FLow User eXperience  -+
Version: 1.0.1
Parsing file: /usr/metron/0.3.0/flux/pcap/remote.yaml
636  [main] INFO  o.a.s.f.p.FluxParser - loading YAML from input stream...
638  [main] INFO  o.a.s.f.p.FluxParser - Performing property substitution.
639  [main] INFO  o.a.s.f.p.FluxParser - Not performing environment variable
 substitution.
907  [main] WARN  o.a.s.f.FluxBuilder - Found multiple invokable methods
 for class org.apache.metron.spout.pcap.SpoutConfig, method from, given
 arguments [END]. Using the last one found.
976  [main] INFO  o.a.s.f.FluxBuilder - Detected DSL topology...
---------- TOPOLOGY DETAILS ----------
Topology Name: pcap
--------------- SPOUTS ---------------
kafkaSpout [1] (org.apache.metron.spout.pcap.KafkaToHDFSSpout)
---------------- BOLTS ----------------
--------------- STREAMS ---------------
---------------------------------------
1157 [main] INFO  o.a.s.f.Flux - Running remotely...
1157 [main] INFO  o.a.s.f.Flux - Deploying topology in an ACTIVE state...
1194 [main] INFO  o.a.s.StormSubmitter - Generated ZooKeeper secret payload
 for MD5-digest: -8340121339010421700:-4824301672672404920
1268 [main] INFO  o.a.s.s.a.AuthUtils - Got AutoCreds []
1343 [main] INFO  o.a.s.StormSubmitter - Uploading topology jar /tmp/
d5f844e8b1a611e6a6d10a0a570e5f4d.jar to assigned location: /data1/hadoop/
storm/nimbus/inbox/stormjar-49aedc3d-a259-409d-a96b-4b615ce07076.jar
1810 [main] INFO  o.a.s.StormSubmitter - Successfully uploaded topology jar
 to assigned location: /data1/hadoop/storm/nimbus/inbox/stormjar-49aedc3d-
a259-409d-a96b-4b615ce07076.jar
1820 [main] INFO  o.a.s.StormSubmitter - Submitting
 topology pcap in distributed mode with conf
 {"topology.workers":1,"storm.zookeeper.topology.auth.scheme":"digest","storm.zookeeper
2004 [main] INFO  o.a.s.StormSubmitter - Finished submitting topology: pcap
```

View pcap Data

To view the pcap data, use the pcap inspector utility, $METRON_HOME/bin/pcap_inspector.sh. This utility enables you to retrieve and view portions of the sequence files which store the pcap data in HDFS.

Procedure

To view pcap data, use the following command:

```
usage: PcapInspector
 -h,--help               Generate Help screen
 -i,--input <SEQ_FILE>   Input sequence file on HDFS
 -n,--num_packets <N>    Number of packets to dump
```

Filtering pcap Data

You can search or filter the pcap data using either a command line tool or a REST API.

Query pcap Data Using the Fixed Filter Option

You can search or filter the PCAP data by the packet header with the fixed filter command line tool.

The packet header filter is be specified via the -pf or --packet_filter options.

The fixed filter option tool is executed by ${metron_home}/bin/pcap_query.sh [fixed|query]

You can filter or query for the following fields in the PCAP data:

- ip_scr_addr
- ip_dst_addr
- ip_src_port
- ip_dst_port
- protocol
- timestamp

Fixed filter options:

```
 -bop,--base_output_path <arg>   Query result output path. Default is
                                 '/tmp'.
 -bp,--base_path <arg>           Base PCAP data path. Default is
                                 '/apps/metron/pcap'.
 -da,--ip_dst_addr <arg>         Destination IP address.
 -df,--date_format <arg>         Date format to use for parsing start_time
                                 and end_time. Default is to use time in
                                 millis since the epoch.
 -dp,--ip_dst_port <arg>         Destination port.
 -pf, --packet_filter <arg>      Packet filter regex
 -et,--end_time <arg>            Packet end time range. Default is current
                                 system time.
 -nr,--num_reducers <arg>        The number of reducers to use.  Default
                                 is 10.
 -h,--help                       Display help.
 -ir,--include_reverse           Indicates if filter should check swapped
                                 src/dest addresses and IPs.
 -p,--protocol <arg>             IP Protocol.
 -rpf                            Maximum number of records per file.
 -sa,--ip_src_addr <arg>         Source IP address.
 -sp,--ip_src_port <arg>         Source port.
```

```
  -st,--start_time <arg>          (required) Packet start time range.
```

Fixed filter examples:

```
$METRON_HOME/bin/pcap_query.sh fixed \
                                     -st "20160617" \
                                     -df "yyyyMMdd" \
                                     -sa 192.168.138.158 \
                                     -da 123.456.789.012 \
                                     -sp 49197 \
                                     -dp 80 \
                                     -p 6
                                     -rpf 500
```

To search for every packet that has an ip_dst_port of 8080 and contains the text "persist", run:

```
$METRON_HOME/bin/pcap_query.sh fixed \
          --ip_dst_port 8080 \
          --packet_filter \
          "\`persist\`" \
          -st "20170425" \
          -df "yyyyMMdd"
```

Query pcap Data Using the Query Filter Option

You can search or filter the PCAP data using a binary regular expression which can be run on the packet payload itself. This query filter option can produce a very large output and create multiple files populating them with the specified number of records and titling them with timestamps.

The query filter option is specified via the BYTEARRAY_MATCHER(pattern, data) Stellar function. The first argument is the regex pattern and the second argument is the data. The packet data will be exposed via the packet variable in Stellar.

The query filter option tool is executed by ${metron_home}/bin/pcap_query.sh [fixed|query].

You can filter or query for the following fields in the PCAP data:

- ip_scr_addr
- ip_dst_addr
- ip_src_port
- ip_dst_port
- protocol
- timestamp

Query filter options:

```
  -bop,--base_output_path <arg>   Query result output path. Default is
                                  '/tmp'.
  -bp,--base_path <arg>           Base PCAP data path. Default is
                                  '/apps/metron/pcap'.
  -df,--date_format <arg>         Date format to use for parsing start_time
                                  and end_time. Default is to use time in
                                  millis since the epoch.
  -et,--end_time <arg>            Packet end time range. Default is current
                                  system time.
  -nr,--num_reducers <arg>        The number of reducers to use.  Default
                                  is 10.
  -h,--help                       Display help.
  -q,--query <arg>                Query string to use as a filter.
  -rpf                            Maximum number of records per file.
  -st,--start_time <arg>          (required) Packet start time range.
```

The Query filter's --query argument specifies the Stellar expression to execute on each packet. To interact with the packet, a few variables are exposed:

- packet : The packet data (a byte[])
- ip_src_addr : The source address for the packet (a String)
- ip_src_port : The source port for the packet (an Integer)
- ip_dst_addr : The destination address for the packet (a String)
- ip_dst_port : The destination port for the packet (an Integer)
- BYTEARRAY_MATCHER : The first argument is the regex pattern and the second argument is the data. The packet data will be exposed via thepacket variable in Stellar.

Query filter examples:

```
$METRON_HOME/bin/pcap_query.sh query \
                                     -st "20160617" \
                                     -df "yyyyMMdd" \
                                     --query "ip_src_addr ==
 '192.168.138.158' and ip_src_port == '49197' \
                                                and ip_dst_addr ==
 '123.456.789.012' and ip_dst_port == '80' \
                                                and protocol == '6'"
                                      -rpf 500
```

To search for every packet that has an ip_dst_port of 8080 and contains the text "persist", run:

```
$METRON_HOME/bin/pcap_query.sh query \
          --query "ip_dst_port == 8080 &&
            BYTEARRAY_MATCHER('\`persist\`', packet)" \
            -st "20170425" \
            -df "yyyyMMdd"
```

You can also do proper binary regexes that look for packets containing the text "persist" and the 2 byte sequence 0x1F909 (in hex):

```
$METRON_HOME/bin/pcap_query.sh query \
          --query "BYTEARRAY_MATCHER('1F90', packet) &&
          BYTEARRAY_MATCHER('\`persist\`', packet)" \
          -st "20170425" \
          -df "yyyyMMdd"
```

Other examples:

```
$METRON_HOME/bin/pcap_query.sh query \
                                     -st "1466136000000" \
                                     --query "IN_SUBNET(ip_src_addr,
 '192.168.0.0/24') and ip_src_port == '49197' \
                                                and ip_dst_addr ==
 '123.456.789.012' and ip_dst_port == '80' \
                                                and protocol == '6'"
                                     -rpf 500
```

```
# subnet function checks IP is in specified subnet
      --query "IN_SUBNET(ip_src_addr, '192.168.0.0/24') \
               and ip_src_port == '49197' \
               and ip_dst_addr == '123.456.789.012' \
               and ip_dst_port == '80' \
               and protocol == '6'"
```

```
# range queries on ports
```

```
      --query "ip_src_port <= 50000 and ip_dst_port >= 30000"
```

```
# range queries with conditionals and parens
      --query "(ip_src_port < 50000 and ip_src_port > 40000) \
            or (ip_src_port < 20000 and ip_src_port > 10000)"
```

```
# in/not in list of values
      --query "ip_src_port < 10000 and ip_dst_port in ['54056', '54057',
 '8080']"
```

Methods to Execute PCAP Filter Options

HCP provides two methods that you can use to query pcap data using the filter options. The two available methods are: PCAP user interface and command line. The PCAP user interface uses the fixed filter option. The CLI method can use either the fixed filter option or the query filter option.

Using the PCAP Panel UI to Query pcap Data

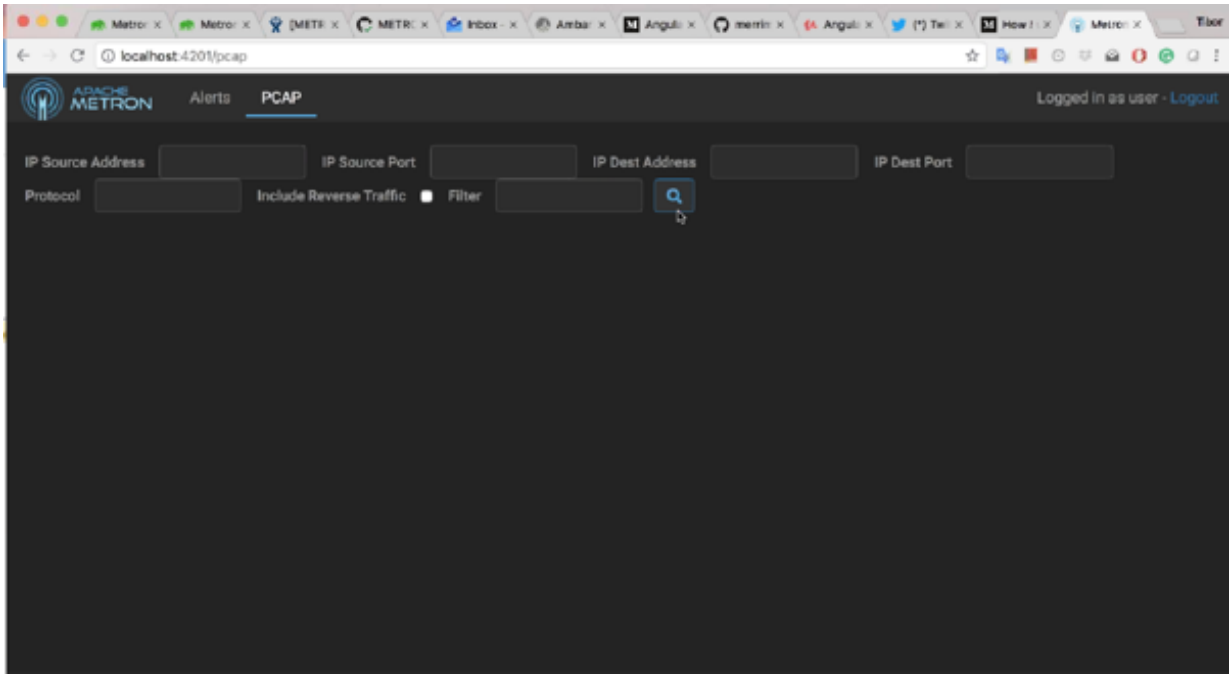
The PCAP panel user interface is ideally suited for the SOC analyst who is identifying and investigating malicious events. You can use the PCAP panel to refine query information provided by other UIs such as the Alerts or Kibana tools. The PCAP panel uses the fixed filter option to query the PCAP data. The PCAP panel provides a graphical user interface to explicitly define the parameters used in the query.

Procedure

1. Display the Ambari user interface.
2. If you have not already done so, in the Services pane, select **Metron**.
3. From the **Quick Links** menu, choose **Alerts UI**.
4. From the Alerts UI, click the **PCAP** tab.
5. Use the date, IP information, and protocol fields to define the parameters of your query, then click  (search icon).

   Note: Only one job can be run at a time.

   You can stop a job by clicking the **X** next to the job progress bar.



Note: The response time of a query is dependent on the precision of your search parameters. If your search parameters are too broad, the query could take a long time and provide results that are too imprecise to be helpful.

From and To	The starting and end dates of your search. The **From** field defaults to 5 days prior to current date. The **To** field defaults to the current date and time.
IP Source Address	The IP source address.
IP Source Port	The IP source port.
IP Dest Address	The IP destination address.
IP Dest Port	The IP destination port.
Protocol	The network protocol. This should be the string value of the protocol.
Include Reverse Traffic	Queries bi-directionally. Runs the query so that it swaps the order of the query between IP Source address and Destination Address.
Filter	Allows you to run a binary regular expression. Filtering can be done both by the packet header as well as with a binary regular expression which can be run on the packet payload itself. This filter can be specified with: • The -pf or --packet_filter options for the fixed query filter • The BYTEARRAY_MATCHER(pattern, data) Stellar function. The first argument is the regex pattern and the second argument is the data. The packet data will be exposed by thepacket variable in Stellar.

6. To download the PCAP filter data, click **Download PCAP**, then specify where to save the data.

Using the CLI to Query pcap Data With the Fixed Filter Option

You can use the CLI to run both types of filter queries. When using the CLI with the fixed filter option, you can utilize more options that you can when using the PCAP panel user interface. For example, the PCAP panel user interface does not have an option to specify the number of reducers to use. The fixed filter filters PCAP data by the packet header. The filter runs on explicit matches only so you cannot use any specialized functions or comparison operators.

Procedure

Execute the fixed filter option using the fixed option:

```
$METRON_HOME/bin/pcap_query.sh fixed
```

You can query for the following fields in the PCAP data:

- ip_scr_addr
- ip_dst_addr
- ip_src_port
- ip_dst_port
- protocol
- timestamp

You can use the following fixed filter options:

-bop,--base_output_path <arg>	Query result output path. Default is /tmp.
-bp,--base_path <arg>	Base PCAP data path. Default is /apps/metron/pcap.
-da,--ip_dst_addr <arg>	Destination IP address.
-df,--date_format <arg>	Date format to use for parsing start_time and end_time. Default is to use time in millis since the epoch.
-dp,--ip_dst_port <arg>	Destination port.
-pf, --packet_filter <arg>	Packet filter regex.
-et,--end_time <arg>	Packet end time range. Default is current system time.
-ft,--finalizer_threads <arg>	Number of threads to use for the final output writing.
-nr,--num_reducers <arg>	The number of reducers to use. Default is 10.
-h,--help	Display help.
-ir,--include_reverse	Indicates if filter should check swapped src/dest addresses and IPs.
-p,--protocol <arg>	IP Protocol.
-rpf,--records_per_file <arg>	Maximum number of records per file.
-sa,--ip_src_addr <arg>	Source IP address.
-sp,--ip_src_port <arg>	Source port.
-st,--start_time <arg>	(required) Packet start time range.
-yq,--yarn_queue <arg>	Yarn queue this job will be submitted to.

For example:

```
$METRON_HOME/bin/pcap_query.sh fixed \
```

```
                                        -st "20160617" \
                                        -df "yyyyMMdd" \
                                        -sa 192.168.138.158 \
                                        -da 123.456.789.012 \
                                        -sp 49197 \
                                        -dp 80 \
                                        -p 6
                                        -rpf 500
```

To search for every packet that has an ip_dst_port of 8080 and contains the text "persist", run:

```
$METRON_HOME/bin/pcap_query.sh fixed \
          --ip_dst_port 8080 \
          --packet_filter \
          "\`persist\`" \
          -st "20170425" \
          -df "yyyyMMdd"
```

Using the CLI to Query pcap Data With the Query Filter Option

Only the CLI enables you to use the query filter option. The query filter leverages Stellar and allows you to more flexibly define the parameters used by the query. This filter option uses a binary regular expression that can be run on the packet payload itself. The query filter option can produce a very large output and create multiple files populating them with the specified number of records and titling them with timestamps.

About this task

The query filter option is specified with the BYTEARRAY_MATCHER(pattern, data) Stellar function. The first argument is the regex pattern and the second argument is the data. The packet data will be exposed with the packet variable in Stellar.

Procedure

To execute the query filter option, run the following:

```
$METRON_HOME/bin/pcap_query.sh query
```

You can filter or query for the following fields in the PCAP data:

- ip_scr_addr
- ip_dst_addr
- ip_src_port
- ip_dst_port
- protocol
- timestamp

The query filter uses the following options:

-bop,--base_output_path <arg>	Query result output path. Default is /tmp.
-bp,--base_path <arg>	Base PCAP data path. Default is /apps/metron/pcap.
-df,--date_format <arg>	Date format to use for parsing start_time and end_time. Default is to use time in millis since the epoch.
-et,--end_time <arg>	Packet end time range. Default is current system time.
-ft,--finalizer_threads <arg>	Number of threads to use for the final output writing.
-nr,--num_reducers <arg>	The number of reducers to use. Default is 10.
-q,--query <arg>	Query string to use as a filter.

-rpf,--records_per_file <arg>	Maximum number of records per file.
-st,--start_time <arg>	(required) Packet start time range.
-yq,--yarn_queue <arg>	Yarn queue this job will be submitted to.

The Query filter's --query argument specifies the Stellar expression to execute on each packet. To interact with the packet, a few variables are exposed:

packet	The packet data (a byte[])
ip_src_addr	The source address for the packet (a String)
ip_src_port	The source port for the packet (an Integer)
ip_dst_addr	The destination address for the packet (a String)
ip_dst_port	The destination port for the packet (an Integer)
BYTEARRAY_MATCHER	The first argument is the regex pattern and the second argument is the data. The packet data will be exposed by thepacket variable in Stellar.

Example

Query filter examples:

```
$METRON_HOME/bin/pcap_query.sh query \
                                      -st "20160617" \
                                      -df "yyyyMMdd" \
                                      --query "ip_src_addr ==
 '192.168.138.158' and ip_src_port == '49197' \
                                                and ip_dst_addr ==
 '123.456.789.012' and ip_dst_port == '80' \
                                                and protocol == '6'"
                                       -rpf 500
```

Example

To search for every packet that has an ip_dst_port of 8080 and contains the text "persist", run:

```
$METRON_HOME/bin/pcap_query.sh query \
          --query "ip_dst_port == 8080 &&
            BYTEARRAY_MATCHER('\`persist\`', packet)" \
            -st "20170425" \
            -df "yyyyMMdd"
```

Example

You can also do proper binary regexes that look for packets containing the text "persist" and the 2 byte sequence 0x1F909 (in hex):

```
$METRON_HOME/bin/pcap_query.sh query \
          --query "BYTEARRAY_MATCHER('1F90', packet) &&
          BYTEARRAY_MATCHER('\`persist\`', packet)" \
          -st "20170425" \
          -df "yyyyMMdd"
```

Example

Other examples:

```
$METRON_HOME/bin/pcap_query.sh query \
                                      -st "1466136000000" \
```

```
                                        --query "IN_SUBNET(ip_src_addr,
 '192.168.0.0/24') and ip_src_port == '49197' \
                                                 and ip_dst_addr ==
 '123.456.789.012' and ip_dst_port == '80' \
                                                 and protocol == '6'"
                                        -rpf 500
```

```
# subnet function checks IP is in specified subnet
      --query "IN_SUBNET(ip_src_addr, '192.168.0.0/24') \
               and ip_src_port == '49197' \
               and ip_dst_addr == '123.456.789.012' \
               and ip_dst_port == '80' \
               and protocol == '6'"
```

```
# range queries on ports
      --query "ip_src_port <= 50000 and ip_dst_port >= 30000"
```

```
# range queries with conditionals and parens
      --query "(ip_src_port < 50000 and ip_src_port > 40000) \
            or (ip_src_port < 20000 and ip_src_port > 10000)"
```

```
# in/not in list of values
      --query "ip_src_port < 10000 and ip_dst_port in ['54056', '54057',
 '8080']"
```

Porting pcap Data to Another Application

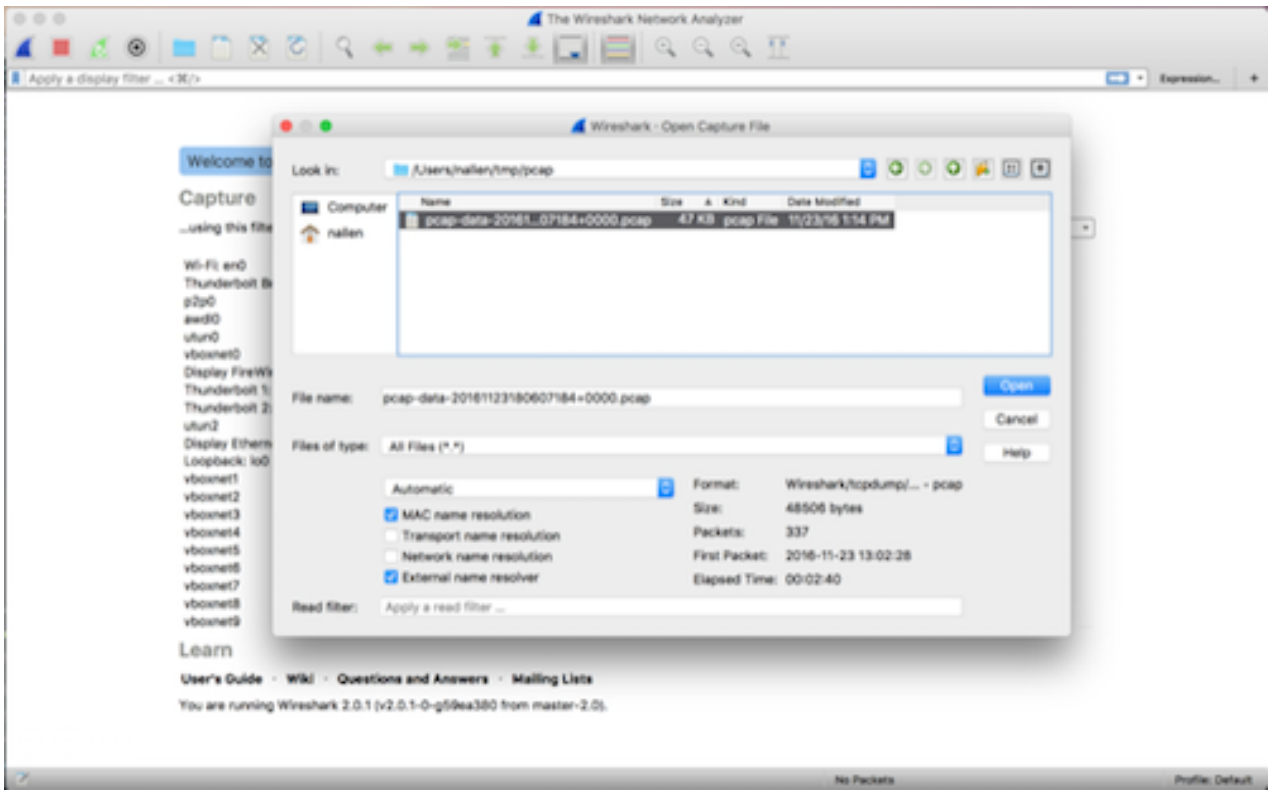
You can port pcap data to another application using the libpcap-compliant pcap file.

When you user the pcap query utility to extract pcap data, the utility creates a libpcap-compliant pcap file in the current working directory.

```
[root@ip-10-0-0-53 0.3.0]# ls -l
total 72
drwxr-xr-x. 2 livy games  4096 Nov 22 22:36 bin
drwxr-xr-x. 3 livy games  4096 Nov 23 17:10 config
drwxr-xr-x. 2 livy games  4096 Sep 29 17:44 ddl
drwxr-xr-x. 6 livy games  4096 Aug 22 14:54 flux
drwxr-xr-x. 2 root root   4096 Nov 23 17:07 lib
drwxr-xr-x. 2 livy games  4096 Nov 22 22:36 patterns
-rw-r--r--. 1 root root  48506 Nov 23 18:06 pcap-
data-20161123180607184+0000.pcap

[root@ip-10-0-0-53 0.3.0]# file pcap-data-20161123180607184+0000.pcap
pcap-data-20161123180607184+0000.pcap: tcpdump capture file (little-endian)
 - version 2.4 (Ethernet, capture length 65535)
```

You can open the libpcap-compliant pcap file with any third-party tool that supports the file type. For example, you can load Wireshark and choose File > Open. Wireshark will load the pcap file.



The content of the file will be similar to the following: