intake_pcap Documentation

Release 0.0.8+0.g220d30a.dirty

Joseph Crail

Apr 25, 2018

Contents:

CHAPTER 1

Quickstart

This guide will show you how to get started using Intake to read packet capture (PCAP) data. It assumes the reader is already familiar with `tcpdump`, the command-line packet analyzer. Given a `tcpdump` command, we will show how you can find the equivalent set of network packets with the Intake PCAP plugin.

1.1 Installation

For conda users, the Intake PCAP plugin is installed with the following commands:

```
conda install -c intake intake-pcap
```

If you wish to follow along with the `tcpdump` examples, consult your OS for the appropriate installation instructions.

1.2 Reading Existing PCAP File

The simplest use case for this plugin is to read an existing PCAP file. Assuming the path to this file is in the variable, `filename`, this will read the entire file into a dataframe.:

```
>>> import intake
>>> ds = intake.open_pcap(filename)
>>> df = ds.read()
>>> df
                        time       src_host src_port       dst_host dst_port protocol
0 2018-01-09 08:16:12.210010   192.168.0.39    54703  172.123.4.567      443      udp
1 2018-01-09 08:16:12.210910   192.168.0.39    54703  172.123.4.567      443      udp
2 2018-01-09 08:16:12.236176  172.123.4.567      443   192.168.0.39    54703      udp
3 2018-01-09 08:16:12.236543  172.123.4.567      443   192.168.0.39    54703      udp
4 2018-01-09 08:16:12.236726   192.168.0.39    54703  172.123.4.567      443      udp
5 2018-01-09 08:16:12.236791   192.168.0.39    54703  172.123.4.567      443      udp
6 2018-01-09 08:16:12.252565  172.123.4.567      443   192.168.0.39    54703      udp
7 2018-01-09 08:16:12.313082  172.123.4.567      443   192.168.0.39    54703      udp
```

(continues on next page)

(continued from previous page)

```
8 2018-01-09 08:16:12.313479  172.123.4.567      443   192.168.0.39    54703      udp
...
```

The remaining sections will describe other use cases. But first, we must setup a sample catalog and its associated data.

1.3 Creating Sample Data

For Unix/Linux users with access to `tcpdump`, you can bootstrap a sample PCAP file with local traffic using the following:

```
sudo tcpdump -c 100 -w local.pcap
```

This will capture 100 packets (including but not exclusive to IP traffic) from the default network interface and write it to a file.

Otherwise, you can use `examples/dump-live.py` to write local traffic to a PCAP file. The syntax for this script is:

```
python examples/dump-live.py PATH INTERFACE LIMIT
```

where `PATH` is the path to a PCAP file, `INTERFACE` is the OS-specific network interface, and `LIMIT` is the number of captured packets.

1.4 Creating a Catalog

The remaining examples assume the existence of a catalog description file, `catalog.yml`, in the same directory as `local.pcap`.:

```
sources:
  raw_live:
    driver: pcap
    args:
      urlpath: ~
      interface: en0
      chunksize: 10
  raw_local:
    driver: pcap
    args:
      urlpath: '{{ CATALOG_DIR }}/local.pcap'
  tcp_local:
    driver: pcap
    args:
      urlpath: '{{ CATALOG_DIR }}/local.pcap'
      protocol: tcp
  udp_local:
    driver: pcap
    args:
      urlpath: '{{ CATALOG_DIR }}/local.pcap'
      protocol: udp
```

This file defines several sources based on the raw sample data we created in the previous section. We will now describe the output associated with each entry.

1.5 Reading a Live Stream

To read a live stream of packets, you will need to start the Python interpreter or Jupyter as a privileged user (`root` on Unix-like systems).

NOTE: Intake does not currently support streaming packets from the network interface. Packets will be placed into a dataframe in chunks (which can be adjusted by the user).

1.5.1 Example: Unfiltered tcpdump

This example will show the first 10 packets on the default interface. Each packet will be timestamped and the raw IP address will be displayed. No packets will be filtered. The exact output will vary depending on your local machine.:

```
$ sudo tcpdump -c 10 -tttt -n -q
2018-01-08 23:37:21.882212 IP 8.8.8.8.53 > 192.168.0.39.61362: UDP, length 172
2018-01-08 23:37:21.882927 IP 192.168.0.39.61447 > 52.12.34.56.443: tcp 0
2018-01-08 23:37:21.953415 IP 52.23.45.67.443 > 192.168.0.39.61445: tcp 0
2018-01-08 23:37:21.953528 IP 192.168.0.39.61445 > 52.23.45.67.443: tcp 0
2018-01-08 23:37:21.991435 IP 52.12.34.56.443 > 192.168.0.39.61447: tcp 0
2018-01-08 23:37:21.991523 IP 192.168.0.39.61447 > 52.12.34.56.443: tcp 0
2018-01-08 23:37:21.993620 IP 192.168.0.39.61447 > 52.12.34.56.443: tcp 517
2018-01-08 23:37:22.093955 IP 52.12.34.56.443 > 192.168.0.39.61447: tcp 0
2018-01-08 23:37:22.099580 IP 52.12.34.56.443 > 192.168.0.39.61447: tcp 1448
2018-01-08 23:37:22.099587 IP 52.12.34.56.443 > 192.168.0.39.61447: tcp 1448
```

1.5.2 Example: Get unfiltered stream of packets without catalog

This example is equivalent to the `tcpdump` example, except the packets will be available in a dataframe. The network interface is required though (typical values are `en0` for macOS and `eth0` for Linux).:

```
>>> import intake
>>> ds = intake.open_pcap(None, interface='en0', chunksize=10)
>>> df = ds.read()
>>> df
                        time      src_host src_port         dst_host dst_port protocol
0 2018-01-09 07:42:36.055605   52.12.34.56      443     192.168.0.39    61614      tcp
1 2018-01-09 07:42:36.055682  192.168.0.39    61614      52.12.34.56      443      tcp
2 2018-01-09 07:42:37.839555  192.168.0.39    17500  255.255.255.255    17500      udp
3 2018-01-09 07:42:37.840472  192.168.0.39    17500    192.168.0.255    17500      udp
4 2018-01-09 07:42:37.890092  192.168.0.39    61614      52.12.34.56      443      tcp
5 2018-01-09 07:42:37.890243  192.168.0.39    61616      52.12.34.56      443      tcp
6 2018-01-09 07:42:37.912166   52.12.34.56      443     192.168.0.39    61616      tcp
7 2018-01-09 07:42:37.912237  192.168.0.39    61616      52.12.34.56      443      tcp
8 2018-01-09 07:42:37.912399  192.168.0.39    61616      52.12.34.56      443      tcp
9 2018-01-09 07:42:37.912833  192.168.0.39    61376     104.12.34.56     4070      tcp
```

1.5.3 Example: Get unfiltered stream of packets with catalog

This example is equivalent to the `tcpdump` example, except the packets will be available in a dataframe. The `raw_live` data source is defined above.:

```
>>> from intake.catalog import Catalog
>>> c = Catalog("catalog.yml")
>>> df = c.raw_live.read()
>>> df
                        time     src_host src_port         dst_host dst_port protocol
0 2018-01-09 07:47:26.825023  192.168.0.1    36123  239.255.255.250     1900      udp
1 2018-01-09 07:47:26.825845  192.168.0.1    36123  239.255.255.250     1900      udp
2 2018-01-09 07:47:26.826602  192.168.0.1    36123  239.255.255.250     1900      udp
3 2018-01-09 07:47:26.827547  192.168.0.1    36123  239.255.255.250     1900      udp
4 2018-01-09 07:47:26.828168  192.168.0.1    36123  239.255.255.250     1900      udp
5 2018-01-09 07:47:26.829162  192.168.0.1    36123  239.255.255.250     1900      udp
6 2018-01-09 07:47:26.829865  192.168.0.1    36123  239.255.255.250     1900      udp
7 2018-01-09 07:47:26.830832  192.168.0.1    36123  239.255.255.250     1900      udp
8 2018-01-09 07:47:26.831615  192.168.0.1    36123  239.255.255.250     1900      udp
9 2018-01-09 07:47:26.832476  192.168.0.1    36123  239.255.255.250     1900      udp
```

1.6 Reading a PCAP File

1.6.1 Example: Unfiltered tcpdump

This example will show the first 10 packets from `local.pcap`. Each packet will be timestamped and the raw IP address will be displayed. No packets will be filtered. The exact output will vary depending on your local machine:

```
$ tcpdump -c 10 -tttt -n -q -r local.pcap
2018-01-09 00:16:12.210010 IP 192.168.0.39.54703 > 172.123.4.567.443: UDP, length 1350
2018-01-09 00:16:12.210910 IP 192.168.0.39.54703 > 172.123.4.567.443: UDP, length 998
2018-01-09 00:16:12.236176 IP 172.123.4.567.443 > 192.168.0.39.54703: UDP, length 1350
2018-01-09 00:16:12.236543 IP 172.123.4.567.443 > 192.168.0.39.54703: UDP, length 31
2018-01-09 00:16:12.236726 IP 192.168.0.39.54703 > 172.123.4.567.443: UDP, length 41
2018-01-09 00:16:12.236791 IP 192.168.0.39.54703 > 172.123.4.567.443: UDP, length 38
2018-01-09 00:16:12.251367 STP 802.1d, Config, Flags [none], bridge-id 7b00.
↪01:23:45:67:89:00.8002, length 35
2018-01-09 00:16:12.252565 IP 172.123.4.567.443 > 192.168.0.39.54703: UDP, length 30
2018-01-09 00:16:12.313082 IP 172.123.4.567.443 > 192.168.0.39.54703: UDP, length 814
2018-01-09 00:16:12.313479 IP 172.123.4.567.443 > 192.168.0.39.54703: UDP, length 16
```

1.6.2 Example: Get unfiltered stream of packets without catalog

This example is equivalent to the `tcpdump` example, except the packets will be available in a dataframe. You should note that there is one less packet in the output since the plugin only shows IP traffic; the `tcpdump` command includes all traffic by default.:

```
>>> import intake
>>> ds = intake.open_pcap("local.pcap")
>>> df = ds.read()
>>> df
                        time       src_host src_port       dst_host dst_port protocol
0 2018-01-09 08:16:12.210010   192.168.0.39    54703  172.123.4.567      443      udp
1 2018-01-09 08:16:12.210910   192.168.0.39    54703  172.123.4.567      443      udp
2 2018-01-09 08:16:12.236176  172.123.4.567      443   192.168.0.39    54703      udp
3 2018-01-09 08:16:12.236543  172.123.4.567      443   192.168.0.39    54703      udp
4 2018-01-09 08:16:12.236726   192.168.0.39    54703  172.123.4.567      443      udp
```

(continues on next page)

(continued from previous page)

```
5 2018-01-09 08:16:12.236791   192.168.0.39    54703  172.123.4.567      443      udp
6 2018-01-09 08:16:12.252565  172.123.4.567      443   192.168.0.39    54703      udp
7 2018-01-09 08:16:12.313082  172.123.4.567      443   192.168.0.39    54703      udp
8 2018-01-09 08:16:12.313479  172.123.4.567      443   192.168.0.39    54703      udp
```

1.6.3 Example: Get unfiltered stream of packets with catalog

This example is equivalent to the `tcpdump` example, except the packets will be available in a dataframe. You should note that there is one less packet in the output since the plugin only shows IP traffic; the `tcpdump` command includes all traffic by default.:

```
>>> from intake.catalog import Catalog
>>> c = Catalog("catalog.yml")
>>> df = c.raw_local.read()
>>> df
                        time       src_host src_port       dst_host dst_port protocol
0 2018-01-09 08:16:12.210010   192.168.0.39    54703  172.123.4.567      443      udp
1 2018-01-09 08:16:12.210910   192.168.0.39    54703  172.123.4.567      443      udp
2 2018-01-09 08:16:12.236176  172.123.4.567      443   192.168.0.39    54703      udp
3 2018-01-09 08:16:12.236543  172.123.4.567      443   192.168.0.39    54703      udp
4 2018-01-09 08:16:12.236726   192.168.0.39    54703  172.123.4.567      443      udp
5 2018-01-09 08:16:12.236791   192.168.0.39    54703  172.123.4.567      443      udp
6 2018-01-09 08:16:12.252565  172.123.4.567      443   192.168.0.39    54703      udp
7 2018-01-09 08:16:12.313082  172.123.4.567      443   192.168.0.39    54703      udp
8 2018-01-09 08:16:12.313479  172.123.4.567      443   192.168.0.39    54703      udp
```

1.7 Filter data

The PCAP plugin will only show IP traffic. If you wish to only see traffic from one protocol, then you can specify one of these values (`tcp`, `udp`, `icmp`, and `igmp`) on the data source.

If you are familiar with the powerful filtering capabilities of `tcpdump`, then you will notice that the plugin's filter is limited at this time.

1.7.1 Example: Get filtered stream of packets without catalog

```
>>> import intake
>>> ds = intake.open_pcap("local.pcap", protocol='udp')
>>> df = ds.read()
>>> df
                        time       src_host src_port       dst_host dst_port protocol
0 2018-01-09 08:16:12.210010   192.168.0.39    54703  172.123.4.567      443      udp
1 2018-01-09 08:16:12.210910   192.168.0.39    54703  172.123.4.567      443      udp
2 2018-01-09 08:16:12.236176  172.123.4.567      443   192.168.0.39    54703      udp
3 2018-01-09 08:16:12.236543  172.123.4.567      443   192.168.0.39    54703      udp
4 2018-01-09 08:16:12.236726   192.168.0.39    54703  172.123.4.567      443      udp
5 2018-01-09 08:16:12.236791   192.168.0.39    54703  172.123.4.567      443      udp
6 2018-01-09 08:16:12.252565  172.123.4.567      443   192.168.0.39    54703      udp
7 2018-01-09 08:16:12.303790  172.123.4.567      443   192.168.0.39    54703      udp
8 2018-01-09 08:16:12.313082  172.123.4.567      443   192.168.0.39    54703      udp
9 2018-01-09 08:16:12.313479  172.123.4.567      443   192.168.0.39    54703      udp
```

1.7.2 Example: Get filtered stream of packets with catalog

```
>>> from intake.catalog import Catalog
>>> c = Catalog("catalog.yml")
>>> df = c.udp_local.read()
>>> df
                        time       src_host src_port       dst_host dst_port protocol
0 2018-01-09 08:16:12.210010   192.168.0.39    54703  172.123.4.567      443      udp
1 2018-01-09 08:16:12.210910   192.168.0.39    54703  172.123.4.567      443      udp
2 2018-01-09 08:16:12.236176  172.123.4.567      443   192.168.0.39    54703      udp
3 2018-01-09 08:16:12.236543  172.123.4.567      443   192.168.0.39    54703      udp
4 2018-01-09 08:16:12.236726   192.168.0.39    54703  172.123.4.567      443      udp
5 2018-01-09 08:16:12.236791   192.168.0.39    54703  172.123.4.567      443      udp
6 2018-01-09 08:16:12.252565  172.123.4.567      443   192.168.0.39    54703      udp
7 2018-01-09 08:16:12.303790  172.123.4.567      443   192.168.0.39    54703      udp
8 2018-01-09 08:16:12.313082  172.123.4.567      443   192.168.0.39    54703      udp
9 2018-01-09 08:16:12.313479  172.123.4.567      443   192.168.0.39    54703      udp
```

1.8 Display packet payload

By default, the full packet data is not included. However, if you wish to see the binary data, then you can set `payload=True` on the data source. For example,:

```
>>> import intake
>>> ds = intake.open_pcap("local.pcap", payload=True)
>>> df = ds.read()
>>> df
                        time       src_host src_port       dst_host dst_port protocol␣
↪ payload
0 2018-01-09 08:16:12.210010   192.168.0.39    54703  172.123.4.567      443      udp␣
↪ j23j4n234023023d
1 2018-01-09 08:16:12.210910   192.168.0.39    54703  172.123.4.567      443      udp␣
↪ df9b9i293ivaiqid
2 2018-01-09 08:16:12.236176  172.123.4.567      443   192.168.0.39    54703      udp␣
↪ j23irg93f9129ed1
3 2018-01-09 08:16:12.236543  172.123.4.567      443   192.168.0.39    54703      udp␣
↪ ni23nf2jg92j3f91
4 2018-01-09 08:16:12.236726   192.168.0.39    54703  172.123.4.567      443      udp␣
↪ 12dj1nd1281j2d12
5 2018-01-09 08:16:12.236791   192.168.0.39    54703  172.123.4.567      443      udp␣
↪ ni12rn30fj9j1j2e
6 2018-01-09 08:16:12.252565  172.123.4.567      443   192.168.0.39    54703      udp␣
↪ 18291n182d12j912
7 2018-01-09 08:16:12.303790  172.123.4.567      443   192.168.0.39    54703      udp␣
↪ 21nd91n2f192fn91
8 2018-01-09 08:16:12.313082  172.123.4.567      443   192.168.0.39    54703      udp␣
↪ n93f293nf2398f23
9 2018-01-09 08:16:12.313479  172.123.4.567      443   192.168.0.39    54703      udp␣
↪ 9tt9090239d903g9
```

CHAPTER 2

API Reference

Plugin()	
intake_pcap.source.PCAPSource(urlpath, . . .)	**Attributes**
intake_pcap.packet.IPPacket(data)	**Attributes**
intake_pcap.stream.LiveStream(interface[, . . .])	**Attributes**
intake_pcap.stream.OfflineStream(path[, . . .])	**Attributes**
intake_pcap.stream.PacketStream(reader, . . .)	**Attributes**

class intake_pcap.**Plugin**

Methods

open(urlpath, **kwargs)	Parameters: urlpath : str Absolute or relative path to source files that can contain shell-style wildcards.

separate_base_kwargs	

open (*urlpath*, ***kwargs*)

Parameters:

urlpath [str] Absolute or relative path to source files that can contain shell-style wildcards.

kwargs [dict] Additional parameters to pass to `intake_pcap.stream.PacketStream` subclass.

class `intake_pcap.source.`**`PCAPSource`**(*urlpath*, *pcap_kwargs*, *metadata*)

Attributes

plot

Methods

`close`()	Close open resources corresponding to this data source.
`discover`()	Open resource and populate the source attributes.
`read`()	Load entire dataset into a container and return it
`read_chunked`()	Return iterator over container fragments of data source
`read_partition`(i)	Return a (offset_tuple, container) corresponding to i-th partition.
`to_dask`()	Return a dask container for this data source

class `intake_pcap.packet.`**`IPPacket`**(*data*)

Attributes

destination_ip_address

destination_ip_port

destination_mac_address

ethernet_protocol

ip_protocol

source_ip_address

source_ip_port

source_mac_address

class `intake_pcap.stream.`**`LiveStream`**(*interface*, *protocol=None*, *payload=False*, *max_packet=65536*, *timeout=1000*)

Attributes

dtype

Methods

`set_filter`(protocol)	Filters all IP traffic except packets matching given protocol.

to_dataframe	

class `intake_pcap.stream.`**`OfflineStream`**(*path*, *protocol=None*, *payload=False*)

Attributes

dtype

Methods

`set_filter`(protocol)	Filters all IP traffic except packets matching given protocol.

to_dataframe	

class `intake_pcap.stream.`**`PacketStream`**(*reader*, *protocol*, *payload*)

Attributes

dtype

Methods

`set_filter`(protocol)	Filters all IP traffic except packets matching given protocol.

to_dataframe	

`set_filter`(*protocol*)

Filters all IP traffic except packets matching given protocol.

Parameters:

protocol [str] Show only traffic for given IP protocol.

Allowed values are icmp, icmp6, igmp, igrp, pim, ah, esp, vrrp, udp, and tcp. If None, all traffic is shown.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Index