chains

Release 0.2.0

Jul 17, 2018

Contents

The Chains project is an exploration of python components that you 'chain' together to process streaming network packets. The use of native python generators means the code is extremely lightweight and efficient.

CHAPTER 1

Install/Run Stuff

Want to see what's happening on your network right now? Just install chains and run 'netwatch'.

```
$ pip install chains
$ netwatch -s
2015-09-07 19:08:34 - UDP IP 192.168.1.9(internal)--> 224.0.0.251(multicast_dns)
2015-09-07 19:08:34 - UDP IP6 fe80::6e40:8ff:fe89:fc08(internal)-->␣
↪ff02::fb(multicast_dns)
2015-09-07 19:08:34 - UDP IP 192.168.1.14(internal)--> 224.0.0.251(multicast_dns)
2015-09-07 19:08:34 - UDP IP6 fe80::8a0:4946:3c8a:e6a1(internal)-->␣
↪ff02::fb(multicast_dns)
2015-09-07 19:08:34 - TCP IP 192.168.1.9(internal)--> 49.75.183.151(nxdomain)
2015-09-07 19:08:36 - TCP IP 192.168.1.9(internal)--> 54.164.252.174(compute-1.
↪amazonaws.com)
2015-09-07 19:08:36 - UDP IP 192.168.1.1(internal)--> 192.168.1.9(internal)
2015-09-07 19:08:36 - TCP IP 54.164.252.174(compute-1.amazonaws.com)--> 192.168.1.
↪9(internal)
...
```

Want to go to coffee shop and see http(s) requests floating about?

```
$ urlwatch

HTTP_REQUEST
192.168.1.9 --> Host: clc.stackoverflow.com
URI: /j/p.js?d=hireme&ac=891012&tags=python;attributes&lw=5913&bw=1539
Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/537.36

HTTP_REQUEST
192.168.1.9 --> Host: ajax.googleapis.com
URI: /ajax/libs/jquery/1.7.1/jquery.min.js
Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/537.36

HTTPS_REQUEST
192.168.1.9 --> 199.166.0.200(sc.iasds01.com) tls_records(5)
```

(continues on next page)

(continued from previous page)

```
TLSRecord(length=512, version=769, type=22, data='\x01\x00\x01\xfc\x03\x03K\t\xf8_\x8.
↪..
TLSRecord(length=560, version=771, type=23, data='\x1d\x942K\xfb\x87\x19v\xba\x13\x14.
↪..
...
```

CHAPTER 2

FAQ

2.1 Can't open network interface

If you get errors about not being able to open the network interface:

- install wireshark and try again (preferred)*
- run the script as sudo

Installing Wireshark changes the permissions of your interfaces so that it can read them. I actually recommend this for several reasons. One it's better than running scripts as sudo and two wireshark is great :)

CHAPTER 3

Examples and References

3.1 Examples

We present a set of examples that hopefully show how you can use Chains to build flexible pipelines of streaming data.

3.1.1 Lets Print some Packets

Printing packets is about the simplest chain you could have. It takes a

- PacketStreamer() `chains.sources.packet_streamer`
- PacketMeta() *chains.links.packet_meta*
- ReverseDNS() `chains.links.reverse_dns`
- PacketPrinter() `chains.sinks.packet_printer`

We link these together in a chain (see what I did there) and we pull the chain. Pulling the chain will stream data from one component to another which only uses the memory required to hold one packet. You could literally run this all day every day for a year on your home network and never run out of memory.

Code from examples/simple_packet_print.py

```
# Create the classes
streamer = packet_streamer.PacketStreamer(iface_name=data_path, max_packets=50)
meta = packet_meta.PacketMeta()
rdns = reverse_dns.ReverseDNS()
printer = packet_printer.PacketPrinter()

# Set up the chain
meta.link(streamer)
rdns.link(meta)
printer.link(rdns)
```

(continues on next page)

(continued from previous page)

```
# Pull the chain
printer.pull()
```

Example Output

```
Timestamp: 2015-05-27 01:17:07.919743
Ethernet Frame: 6c:40:08:89:fc:08 --> 01:00:5e:00:00:fb  (type: 2048)
Packet: IP 192.168.1.9 --> 224.0.0.251 (len:55 ttl:255) -- Frag(df:0 mf:0 offset:0)
Domains: LOCAL --> multicast_dns
Transport: UDP {'dport': 5353, 'sum': 59346, 'sport': 5353, 'data':
↪'\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x03CTV\x05local\x00\x00\x1c\x80\x01
↪', 'ulen': 35}
Application: None

Timestamp: 2015-05-27 01:17:07.919926
Ethernet Frame: 6c:40:08:89:fc:08 --> 33:33:00:00:00:fb  (type: 34525)
Packet: IP6 fe80::6e40:8ff:fe89:fc08 --> ff02::fb (len:35 ttl:255)
Domains: LOCAL --> multicast_dns
Transport: UDP {'dport': 5353, 'sum': 6703, 'sport': 5353, 'data':
↪'\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x03CTV\x05local\x00\x00\x1c\x80\x01
↪', 'ulen': 35}
Application: None
...
```

3.1.2 Taggers

Taggers can look at the streaming data and add tags

- PacketStreamer() `chains.sources.packet_streamer`
- PacketMeta() *chains.links.packet_meta*
- ReverseDNS() `chains.links.reverse_dns`
- Tagger() `chains.links.tagger`
- PacketPrinter() `chains.sinks.packet_printer`

Again we simply link these together in a chain and then pull the chain.

Code from examples/tag_example.py

```
# Create the classes
streamer = packet_streamer.PacketStreamer(iface_name=data_path, max_packets=50)
meta = packet_meta.PacketMeta()
rdns = reverse_dns.ReverseDNS()
tags = tagger.Tagger()
printer = packet_summary.PacketSummary()

# Set up the chain
meta.link(streamer)
rdns.link(meta)
tags.link(rdns)
printer.link(tags)

# Pull the chain
printer.pull()
```

Example Output

```
2015-05-30 00:34:45 - TCP IP 192.168.1.9(LOCAL) --> 12.226.156.82(NXDOMAIN) TAGS: [
↪'outgoing', 'nxdomain']
2015-05-30 00:34:45 - TCP IP 12.226.156.82(NXDOMAIN) --> 192.168.1.9(LOCAL) TAGS: [
↪'incoming', 'nxdomain']
2015-05-30 00:34:45 - TCP IP 192.168.1.9(LOCAL) --> 54.197.119.105(compute-1.
↪amazonaws.com) TAGS: ['outgoing']
...
```

3.2 Class Reference

The Chains packets is split into following sub-packages

- Sources: Classes with produce data (they have an output_stream but NOT an input_stream)
- Links: Classes that consume an input_stream and produce an output_stream
- Sinks: Classes that consume an input_stream but do NOT produce an output_stream
- Utils: Helpful utility methods (HHAC/IP to str, file helpers, log utils)

3.2.1 Sources

Sources are classes with produce data (they have an output_stream but NOT an input_stream)

PacketStreamer

Source BaseClass

Sources provide an output_stream but do not take in input stream

class chains.sources.source.**Source**
Bases: *chains.links.link.Link*

Sources provide an output_stream but do not take in input stream

input_stream
The input stream property (not provided for a source)

chains.sources.source.**test**()
Spin up the source class and call the methods

3.2.2 Links

Links are classes that consume an input_stream and produce an output_stream.

PacketMeta

PacketMeta, Use DPKT to pull out packet information and convert those attributes to a dictionary based output.

class chains.links.packet_meta.**PacketMeta**
Bases: *chains.links.link.Link*

PacketMeta, Use DPKT to pull out packet information and convert those attributes to a dictionary based output.

packet_meta_data()
Pull out the metadata about each packet from the input_stream :param None:

Returns a generator that contains packet meta data in the form of a dictionary

Return type generator (dictionary)

chains.links.packet_meta.**test**()
Test for PacketMeta class

ReverseDNS

PacketTags

TransportMeta

TransportMeta: Pull out transport meta data from incoming packet data

class chains.links.transport_meta.**TransportMeta**
Bases: *chains.links.link.Link*

Pull out transport meta data from incoming packet data

transport_meta_data()
Pull out the transport metadata for each packet in the input_stream

chains.links.transport_meta.**test**()
Test for TransportMeta class

Flows

HTTPMeta

HTTPMeta: Pull out HTTP meta data from incoming flow data

class chains.links.http_meta.**HTTPMeta**
Bases: *chains.links.link.Link*

Pull out application meta data from incoming flow data

http_meta_data()
Pull out the application metadata for each flow in the input_stream

chains.links.http_meta.**test**()
Test for HTTPMeta class

Link BaseClass

Links take an input_stream and provides an output_stream. All streams are required to be a generator that yields python dictionaries.

class chains.links.link.**Link**
Bases: object

Link classes take an input_stream and provide an output_stream. All streams are required to be a generator that yields python dictionaries.

link(*stream_instance*)
Set my input stream

input_stream
The input stream property

output_stream
The output stream property

chains.links.link.**test**()
Spin up the link class and call the methods

3.2.3 Sinks

Sinks are casses that consume an input_stream but do NOT produce an output_stream.

PacketPrinter

PacketSummary

Sink BaseClass

Sinks take an input_stream and provides a pull() method. Note: Sinks do not provide an output_stream.

class chains.sinks.sink.**Sink**
Bases: *chains.links.link.Link*

Sinks take an input_stream and provides a pull() method Note: Sinks do not provide an output_stream.

output_stream
The output stream property

pull()
Process the input stream

chains.sinks.sink.**test**()
Spin up the sink class and call the methods

3.2.4 Utils

Just a set of helpful utility methods (HHAC/IP to str, file helpers, log utils).

Network Utils

File Utils

File utilities that might be useful

chains.utils.file_utils.**all_files_in_directory**(*path*)
Recursively list all files under a directory

Parameters **path** – the path of the directory to traverse

Returns a list of all the files contained withint the directory

chains.utils.file_utils.**file_dir**(*file_path*)
Root directory for a file_path

Parameters **file_path** – a fully qualified file path

Returns the directory which contains the file

chains.utils.file_utils.**relative_dir**(*file_path*, *rel_dir*)
Relative directory to the file_path

Parameters **file_path** – a fully qualified file path

Returns the relative directory

chains.utils.file_utils.**test_utils**()
Test the utility methods

Log Utils

Log utilities that might be useful

chains.utils.log_utils.**get_logger**()
Setup logging output defaults

chains.utils.log_utils.**panic**(*mesg*)
Throw a critical message and raise a runtime exceptions

chains.utils.log_utils.**test_utils**()
Test the utility methods

Data Utils

Data utilities that might be useful

chains.utils.data_utils.**make_dict**(*obj*)
This method creates a dictionary out of a non-builtin object

chains.utils.data_utils.**is_builtin**(*obj*)

chains.utils.data_utils.**get_value**(*data*, *key*)
Follow the dot notation to get the proper field, then perform the action

Parameters

- **data** – the data as a dictionary (required to be a dictionary)
- **key** – the key (as dot notation) into the data that gives the field (IP.src)

Returns: the value of the field(subfield) if it exist, otherwise None

chains.utils.data_utils.**test_utils**()
Test the utility methods

Cache

Cache class for key/value pairs

class chains.utils.cache.**Cache**(*max_size=1000*, *timeout=None*)
Bases: object

In process memory cache. Not thread safe. Usage:

cache = Cache(max_size=5, timeout=10) cache.set('foo', 'bar') cache.get('foo') >>> bar time.sleep(11) cache.get('foo') >>> None cache.clear()

set(*key*, *value*)
Add an item to the cache :param key: item key :param value: the value associated with this key

get(*key*)
Get an item from the cache :param key: item key

Returns the value of the item or None if the item isn't in the cache

clear()
Clear the cache

dump()
Dump the cache (for debugging)

chains.utils.cache.**test**()
Test for the Cache class

CHAPTER 4

Help the Project

4.1 Contributing

4.1.1 Report a Bug or Make a Feature Request

Please go to the GitHub Issues page: https://github.com/supercowpowers/chains/issues.

4.1.2 Checkout the Code

```
git clone https://github.com/supercowpowers/chains.git
```

4.1.3 Become a Developer

chains uses the 'GitHub Flow' model: GitHub Flow

- To work on something new, create a descriptively named branch off of master (ie: my-awesome)
- Commit to that branch locally and regularly push your work to the same named branch on the server
- When you need feedback or help, or you think the branch is ready for merging, open a pull request
- After someone else has reviewed and signed off on the feature, you can merge it into master

New Feature or Bug

```
$ git checkout -b my-awesome
$ git push -u origin my-awesome
$ <code for a bit>; git push
$ <code for a bit>; git push
$ tox (this will run all the tests)
```

- Go to github and hit ‘New pull request’
- Someone reviews it and says ‘AOK’
- Merge the pull request (green button)

CHAPTER 5

Indices and tables

- genindex
- modindex

Python Module Index

l

s

u

Index