AntiNex Network Pipeline Documentation

Release 1.0.0

Jay Johnson

Nov 02, 2018

Contents

CHAPTER 1

AntiNex Stack Status

AntiNex Network Pipeline is part of the AntiNex stack:

Component	Build	Docs Link	Docs Build
REST API		Docs	
Core Worker		Docs	
Network Pipeline		Docs	
AI Utils		Docs	
Client		Docs	

CHAPTER 2

Table of Contents

These are the docs for the AntiNex Network Pipeline repository.

2.1 Source Code

2.1.1 Handle Packets from a Network Interface

This is the default handler for processing network packets received from the network interface with `eth0` or `eth1`. In production, this is the starting point for making live predictions with the AntiNex REST API.

Here is the workflow for processing a network packet from a monitored interface:

1. Get Available Layers in the Packet
2. Convert the Packet to a JSON dictionary
3. Publish the Message using Kombu with environment values setting the routing decision for the message in the aggregation message broker: `FORWARD_EXCHANGE`, `FORWARD_ROUTING_KEY`, `FORWARD_QUEUE`.

`network_pipeline.handle_packets.`**`handle_packets`**(*pk*)

Parameters **pk** – data packet that kamene sends in

2.1.2 Process Consumed Messages from the Queue

This is the default handler for processing messages consumed from the aggregration message broker. At the conceptual level, all network interface capture tools forward JSON dictionaries to this class.

class `network_pipeline.record_packets_to_csv.`**`RecordPacketsToCSV`**

build_all_keys_dict()

build_flat_msg(*id=None*, *msg=None*)

Parameters

- **id** – unique id for this message
- **msg** – message dictionary to flatten

convert_to_df()

create_json_archive()

flatten_all()

handle_msg(*body*, *org_message*)

Parameters

- **body** – dictionary contents from the message body
- **org_message** – message object can ack, requeue or reject

process_arp_frame(*id=None*, *msg=None*)

Convert a complex nested json dictionary to a flattened dictionary and capture all unique keys for table construction

Parameters

- **id** – key for this msg
- **msg** – arp frame for packet

process_dns_frame(*id=None*, *msg=None*)

Convert a complex nested json dictionary to a flattened dictionary and capture all unique keys for table construction

Parameters

- **id** – key for this msg
- **msg** – dns frame for packet

process_ether_frame(*id=None*, *msg=None*)

Convert a complex nested json dictionary to a flattened dictionary and capture all unique keys for table construction

Parameters

- **id** – key for this msg
- **msg** – ether frame for packet

process_icmp_frame(*id=None*, *msg=None*)

Convert a complex nested json dictionary to a flattened dictionary and capture all unique keys for table construction

Parameters

- **id** – key for this msg
- **msg** – icmp frame for packet

process_ip_frame(*id=None*, *msg=None*)

Convert a complex nested json dictionary to a flattened dictionary and capture all unique keys for table construction

Parameters

- **id** – key for this msg

- **msg** – ip frame for packet

process_ipvsix_frame(*id=None*, *msg=None*)
Convert a complex nested json dictionary to a flattened dictionary and capture all unique keys for table construction

Parameters

- **id** – key for this msg
- **msg** – ipv6 frame for packet

process_pad_frame(*id=None*, *msg=None*)
Convert a complex nested json dictionary to a flattened dictionary and capture all unique keys for table construction

Parameters

- **id** – key for this msg
- **msg** – pad frame for packet

process_raw_frame(*id=None*, *msg=None*)
Convert a complex nested json dictionary to a flattened dictionary and capture all unique keys for table construction

Parameters

- **id** – key for this msg
- **msg** – raw frame for packet

process_tcp_frame(*id=None*, *msg=None*)
Convert a complex nested json dictionary to a flattened dictionary and capture all unique keys for table construction

Parameters

- **id** – key for this msg
- **msg** – tcp frame for packet

process_udp_frame(*id=None*, *msg=None*)
Convert a complex nested json dictionary to a flattened dictionary and capture all unique keys for table construction

Parameters

- **id** – key for this msg
- **msg** – udp frame for packet

publish_predictions_to_core()

save_data()

save_df_as_csv()

write_to_file(*data_dict*, *output_file_path*)

Parameters

- **data_dict** –
- **output_file_path** –

2.1.3 Network Pipeline Internal Modules

network_pipeline.build_packet_key.**build_packet_key**()

network_pipeline.connect_forwarder.**connect_forwarder**(*forward_host=None*, *forward_port=None*, *max_retries=-1*, *sleep_interval=1.0*)

Parameters

- **forward_host** – host for receiving forwarded packets
- **forward_port** – port for the forwarded packets
- **max_retries** – retries, -1 = infinite
- **sleep_interval** – how often to retry in this loop

network_pipeline.convert_pkt_to_json.**convert_pkt_to_json**(*pkg*)

Inspired by: https://gist.githubusercontent.com/cr0hn/1b0c2e672cd0721d3a07/raw/9144676ceb12dbd545e6dce366822bbedde8de2c/pkg_to_json.py This function convert a Scapy packet to JSON

Parameters **pkg** (*objects*) – A kamene package

Returns A JSON data

Return type dict()

network_pipeline.create_layer_2_socket.**create_layer_2_socket**()

network_pipeline.parse_network_data.**eth_addr**(*f*)

Parameters **f** – eth frame

network_pipeline.parse_network_data.**unshift_flags**(*tcp_flags*)

De-shift the TCP flags to a string repr

network_pipeline.parse_network_data.**build_key**()

network_pipeline.parse_network_data.**parse_network_data**(*data_packet=None*, *include_filter_key=None*, *filter_keys=[]*, *record_tcp=True*, *record_udp=True*, *record_arp=True*, *record_icmp=True*)

build_node

Parameters

- **data_packet** – raw recvfrom data
- **filter_keys** – list of strings to filter and remove baby-birding packets to yourself
- **record_tcp** – want to record TCP frames?
- **record_udp** – want to record UDP frames?
- **record_arp** – want to record ARP frames?
- **record_icmp** – want to record ICMP frames?

network_pipeline.publisher.**get_publisher**()

network_pipeline.utils.**rnow**(*f='%Y-%m-%d %H:%M:%S'*)

Parameters **f** – format for the string

network_pipeline.utils.**ppj**(*json_data*)

Parameters **json_data** – dictionary to print

network_pipeline.start_consumers_for_queue.**start_consumers_for_queue**(*prefix_name='worker'*, *num_workers=2*, *tasks=None*, *queue_to_consume=None*, *shutdown_msg='SHUTDOWN'*, *consumer_class=None*, *need_response=False*, *callback=None*)

Parameters

- **prefix_name** –
- **num_workers** –
- **tasks** –
- **queue_to_consume** –
- **shutdown_msg** –
- **consumer_class** –
- **need_response** –
- **callback** –

class network_pipeline.network_packet_task.**NetworkPacketTask**(*source='locahost'*, *payload=None*)

network_pipeline.shutdown_consumers.**shutdown_consumers**(*num_workers=2*, *tasks=None*, *shutdown_msg='SHUTDOWN'*)

Parameters

- **num_workers** –
- **tasks** –
- **shutdown_msg** –

class network_pipeline.simulated_work_task.**SimulatedWorkTask**(*a*, *b*)

class network_pipeline.worker_to_process_packets.**WorkerToProcessPackets**(*name*, *task_queue*, *result_queue*, *shutdown_msg='SHUTDOWN'*, *need_response=False*, *callback=None*)

run()

2.1.4 Network Pipeline Scripts

Capture Agents

Here are the AntiNex Network Pipeline Capture Agents. These scripts allow for capturing traffic on a network device and flattening it into JSON dictionaries before publishing to the aggregation message broker. Please refer to the `handle_packets` method for more details.

> **Warning:** These tools will capture network traffic. Please be careful where you deploy them.

ARP

`network_pipeline.scripts.capture_arp.`**`capture_arp_packets`**`()`
Capture `ARP` packets and call the `handle_packets` method

Change the network interface by `export CAP_DEVICE=eth0`

ICMP

`network_pipeline.scripts.capture_icmp.`**`capture_icmp_packets`**`()`
Capture `ICMP` packets and call the `handle_packets` method

Change the network interface by `export CAP_DEVICE=eth0`

TCP

`network_pipeline.scripts.capture_ssh.`**`capture_tcp_packets_over_ssh`**`()`
Capture `TCP` packets over ssh and call the `handle_packets` method

Change the network interface by `export CAP_DEVICE=eth0`

`network_pipeline.scripts.capture_tcp.`**`capture_tcp_packets`**`()`
Capture `TCP` packets and call the `handle_packets` method

Change the network interface by `export CAP_DEVICE=eth0`

`network_pipeline.scripts.capture_telnet.`**`capture_tcp_packets_over_telnet`**`()`
Capture `TCP` packets over telnet and call the `handle_packets` method

Change the network interface by `export CAP_DEVICE=eth0`

UDP

`network_pipeline.scripts.capture_udp.`**`capture_udp_packets`**`()`
Capture `UDP` packets and call the `handle_packets` method

Change the network interface by `export CAP_DEVICE=eth0`

Publishers

These tools are designed to show how to save captured packet dictionaries to CSVs and how to publish them for live predictions using a pre-trained Deep Neural Network.

network_pipeline.scripts.packets_rabbitmq.**recv_msg**(*body*, *message*)
Handler method - fires when a messages is consumed from the FORWARD_QUEUE queue running in the FORWARD_BROKER_URL broker.

Parameters

- **body** – message body
- **message** – message object can ack, requeue or reject

network_pipeline.scripts.packets_rabbitmq.**consume_network_packet_messages_from_rabbitmq**()
Setup a celery_connectors.KombuSubscriber to consume meessages from the FORWARD_BROKER_URL broker in the FORWARD_QUEUE queue.

network_pipeline.scripts.packets_redis.**recv_msg**(*body*, *message*)
Handler method - fires when a messages is consumed from the FORWARD_QUEUE queue running in the FORWARD_BROKER_URL broker.

Parameters

- **body** – message body
- **message** – message object can ack, requeue or reject

network_pipeline.scripts.packets_redis.**consume_network_packet_messages_from_redis**()
Setup a celery_connectors.KombuSubscriber to consume meessages from the FORWARD_BROKER_URL broker in the FORWARD_QUEUE queue.

Test Tools

These will send mock traffic data to the targeted network device.

network_pipeline.scripts.base_capture.**example_capture**()
An example capture script

Change the network interface by export CAP_DEVICE=eth0

network_pipeline.scripts.arp_send_msg.**send_arp_msg**()
Send an ARP message to the network device (enp0s3 by default).

network_pipeline.scripts.tcp_send_large_msg.**send_tcp_large_message**()
Send a large TCP message to port 80 by default.

network_pipeline.scripts.tcp_send_msg.**send_tcp_message**()
Send a TCP message to port 80 by default.

network_pipeline.scripts.udp_send_msg.**send_udp_message**()
Send a UDP message to port 80 by default.

Environment variables:

UDP_SEND_TO_HOST - host ip address UDP_SEND_TO_PORT - send to this UDP port

network_pipeline.scripts.listen_tcp_port.**listen_on_tcp_port**()
Run a simple server for processing messages over TCP.

LISTEN_ON_HOST - listen on this host ip address

LISTEN_ON_PORT - listen on this TCP port

LISTEN_SIZE - listen on to packets of this size

LISTEN_SLEEP - sleep this number of seconds per loop

LISTEN_SHUTDOWN_HOOK - shutdown if file is found on disk

network_pipeline.scripts.listen_udp_port.**listen_on_udp_port**()
Run a simple server for processing messages over UDP.

UDP_LISTEN_ON_HOST - listen on this host ip address

UDP_LISTEN_ON_PORT - listen on this UDP port

UDP_LISTEN_SIZE - listen on to packets of this size

UDP_LISTEN_SLEEP - sleep this number of seconds per loop

UDP_LISTEN_SHUTDOWN_HOOK - shutdown if file is found on disk

network_pipeline.scripts.builders.prepare_dataset.**find_all_headers**(*pipeline_files=[]*, *label_rules=None*)

Parameters

- **pipeline_files** – files to process
- **label_rules** – labeling rules

network_pipeline.scripts.builders.prepare_dataset.**build_csv**(*pipeline_files=[]*, *fulldata_file=None*, *clean_file=None*, *post_proc_rules=None*, *label_rules=None*, *metadata_filename='metadata.json'*)

Parameters

- **pipeline_files** – files to process
- **fulldata_file** – output all columns to this csv file
- **clean_file** – output all numeric-ready columns to this csv file
- **post_proc_rules** – rules after building the DataFrame
- **label_rules** – labeling rules
- **metadata_filename** – metadata

network_pipeline.scripts.builders.prepare_dataset.**find_all_pipeline_csvs**(*csv_glob_path='/opt/anti*

Parameters **csv_glob_path** – path to csvs

network_pipeline.scripts.builders.prepare_dataset.**prepare_new_dataset**()

class network_pipeline.scripts.tools.arp_send_msg.**Ethernet**
Generic Ethernet Frame class

class network_pipeline.scripts.tools.arp_send_msg.**Arp**
Generic ARP Frame class

2.1.5 Constants

```
VALID = 0
FILTERED = 1
INVALID = 2
ERROR = 3
UNSUPPORTED = 4
ETH_UNSUPPORTED = 5
IP_UNSUPPORTED = 6

INCLUDED_IGNORE_KEY = "CHANGE_TO_YOUR_OWN_KEY"

ETH_HEADER_FORMAT = "!6s6sH"
IP_HEADER_FORMAT = "!BBHHHBBH4s4s"
TCP_HEADER_FORMAT = "!HHLLBBHHH"
TCP_PSH_FORMAT = "!4s4sBBH"
UDP_HEADER_FORMAT = "!HHHH"
ICMP_HEADER_FORMAT = "!BBH"
ARP_HEADER_FORMAT = "2s2s1s1s2s6s4s6s4s"

SIZE_ETH_HEADER = struct.calcsize(ETH_HEADER_FORMAT)
SIZE_IP_HEADER = struct.calcsize(IP_HEADER_FORMAT)
SIZE_TCP_HEADER = struct.calcsize(TCP_HEADER_FORMAT)
SIZE_UDP_HEADER = struct.calcsize(UDP_HEADER_FORMAT)
SIZE_ICMP_HEADER = struct.calcsize(ICMP_HEADER_FORMAT)
SIZE_ARP_HEADER = struct.calcsize(ARP_HEADER_FORMAT)

UNKNOWN = 0
TCP = 1
UDP = 2
ICMP = 3
ARP = 4

ARP_PROTO_ETH = 9731
ICMP_PROTO_IP = 1
IP_PROTO_ETH = 8
TCP_PROTO_IP = 6
UDP_PROTO_IP = 17

IGNORED_REDIS_PORTS = [6379, 16379]
IGNORED_RABBITMQ_PORTS = [5672, 15672, 25672]
```

2.1.6 Environment Variables

```
SOURCE = os.getenv(
        "SOURCE_HOST",
        "localdev").strip().lstrip()
FORWARD_BROKER_URL = os.getenv(
        "FORWARD_BROKER_URL",
        "redis://localhost:6379/15").strip().lstrip()
FORWARD_SSL_OPTIONS = json.loads(os.getenv(
        "FORWARD_SSL_OPTIONS",
        "{}").strip().lstrip())
FORWARD_ENDPOINT_TYPE = os.getenv(
        "FORMAT_ET",
```

(continues on next page)

(continued from previous page)

```
        "redis").strip().strip()
FORWARD_EXCHANGE = os.getenv(
        "FORWARD_EXCHANGE",
        "NEW_PACKETS").strip().lstrip()
FORWARD_ROUTING_KEY = os.getenv(
        "FORWARD_ROUTING_KEY",
        "NEW_PACKETS").strip().lstrip()
FORWARD_QUEUE = os.getenv(
        "FORWARD_QUEUE",
        "NEW_PACKETS").strip().lstrip()
DEBUG_PACKETS = bool(os.getenv(
        "DEBUG_PACKETS",
        "0").strip().lstrip() == "1")
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

n

Index

L

N

P

R

S

U

W