vespa Documentation

Release 1.0

Aurélien Wailly

December 16, 2014

Contents

Contents:

CHAPTER 1

vespa package

1.1 Submodules

1.2 vespa.aes_gcm module

class vespa.aes_gcm.**AES_GCM**(*master_key*)

change_key(*master_key*)

decrypt(*init_value*, *ciphertext*, *auth_tag*, *auth_data=''*)

encrypt(*init_value*, *plaintext*, *auth_data=''*)

exception vespa.aes_gcm.**InvalidInputException**(*msg*)
Bases: exceptions.Exception

exception vespa.aes_gcm.**InvalidTagException**
Bases: exceptions.Exception

vespa.aes_gcm.**gf_2_128_mul**(*x*, *y*)

1.3 vespa.agent module

Agent representation

class vespa.agent.**Agent**(*name*, *host*, *port*, *master*, *run=True*)
Bases: vespa.node.Node

1.4 vespa.agent_av module

Agent representation

class vespa.agent_av.**Agent_AV**(*name*, *host*, *port*, *master*, *vm*)
Bases: vespa.agent.Agent

Create an Agent able to communicate with the ClamAV backend (need a driver).

Returns The Agent instance to offer the ClamAV support

Return type Node

connect_warning()
Set up the agent for interactions with the VM

dump_analyzed_file_list()
Gather list of files analyzed bi the ClamAV antivirus

Returns The list of analyzed files

Return type list

isolate_warning(*vm*)
Set up the agent for interactions with the hypervisor

Parameters **vm** (*str*) – The tuple (name, host, port) describing the backend

send(*msg*)
Overload the internal send to capture and send messages to the backend

Parameters **msg** (*str*) – The massage to process and to send

Returns The backend response

Return type str

1.5 vespa.agent_bandwidth module

Agent wrapper around /proc/dev/net to filter an interface statistics. The interface *eth0* is used as default.

class vespa.agent_bandwidth.**Agent_Bandwidth**(*name*, *host*, *port*, *master*, *run=True*)
Bases: vespa.agent.Agent

Provide a wrapper around Linux interfaces /proc files. The Agent can extract information of specific interfaces, i.e. eth0 or lo.

Returns The agent to grab informations

Return type Node

get_mac()
Grab the mac address of the class defined _self.iface_

Returns The string containing the mac address, colon separated

Return type str

launch()
Send _recv_bytes_ and _trans_bytes_ back to the master every second

1.6 vespa.agent_connections module

Counting connections as suggested into: http://www.linuxjournal.com/content/back-dead-simple-bash-complex-ddos

SynFlood https://raw.github.com/arthurnn/SynFlood/master/synflood

class vespa.agent_connections.**Agent_Connections**(*name*, *host*, *port*, *master*, *run=True*)
Bases: vespa.agent.Agent

An agent gathering network links through psutil python module or system lsof command

Returns The wrapper

Return type Node

launch()
Return network connections to orchestrator layer every second using either psutil or lsof

1.7 vespa.agent_controller module

Agent to wrap Gandalf's controller

class vespa.agent_controller.**Agent_Controller**(*name*, *host*, *port*, *master*, *run=False*)
Bases: vespa.agent.Agent

Create an Agent to send a mac address to an OpenFlow controller

Returns The Agent instance to offer the OpenFlow alert_ip function

Return type Node

alert_ip(*ip*, *mac*)
Block the mac address on the network

Parameters

- **ip** (*str*) – The IP address or domain of the controller
- **mac** (*str*) – The mac address to block on the network

Returns The "Ok" string

Return type str

1.8 vespa.agent_controller_floodlight module

Agent to wrap Gandalf's controller. Based on floodlight, it can be a nice start for a full API against floodlight.

class vespa.agent_controller_floodlight.**Agent_Controller_Floodlight**(*name*, *host*, *port*, *master*, *run=False*)
Bases: vespa.agent_controller.Agent_Controller

Flag a mac address as suspicious and gather statistics for local links

Returns The wrapper to the OMN controller

Return type Node

alert_ip(*ip*, *mac*)
Block a tuple (ip,mac) with SDN

Parameters

- **IP** (*str*) – The IP to block (for future)
- **mac** (*str*) – The associated MAC address (needed)

Returns The controller response

Return type str

block_hackers(*mac*)
Block a MAC address with SDN

Parameters **mac** (*str*) – The associated MAC address

Returns The controller response

Return type str

get_link_stats(*cmd='wm/topology/links/json'*)
Get links statistics over the floodlight controller

Parameters **cmd** (*str*) – The floodlight URL to grab the links statistics

Returns The controller response

Return type str

get_topology(*cmd='wm/topology/switchclusters/json'*)
Get the current topology of the SDN network

Parameters **cmd** (*str*) – The floodlight URL to grab the topology

Returns The list of nodes and links detected

Return type dict

release_hackers()
Release all tuples (ip,mac) with SDN

Returns The controller response

Return type str

status_hackers()
Get the status of a tuple (ip,mac) with SDN

Returns The controller response

Return type str

1.9 vespa.agent_controller_pox module

Agent to wrap the POX python SDN controller. It require some modification on the other side too. You can follow the mac address blocking tutorial on the POX website.

class vespa.agent_controller_pox.**Agent_Controller_Pox**(*name*, *host*, *port*, *master*, *run=False*)
Bases: vespa.agent_controller.Agent_Controller

Flag a mac address as suspicious and gather statistics for local links

Returns The wrapper to the OMN controller

Return type Node

alert_ip(*ip*, *mac*)
Block a tuple (ip,mac) with SDN

Parameters

- **IP** (*str*) – The IP to block (for future)
- **mac** (*str*) – The associated MAC address (needed)

Returns The controller response

Return type str

block_hackers()
Block a MAC address with SDN

Parameters **mac** (*str*) – The associated MAC address

Returns The controller response

Return type str

get_link_stats(*cmd='get_link_stats'*)
Get links statistics over the pox controller

Parameters **cmd** (*str*) – The POX URL to grab the links statistics

Returns The controller response

Return type str

get_topology(*cmd='get_topology'*)
Get the current topology of the SDN network

Parameters **cmd** (*str*) – The POX URL to grab the topology

Returns The list of nodes and links detected

Return type dict

release_hackers()
Release all tuples (ip,mac) with SDN

Returns The controller response

Return type str

status_hackers()
Get the status of a tuple (ip,mac) with SDN

Returns The controller response

Return type str

1.10 vespa.agent_libvirt module

class vespa.agent_libvirt.**Agent_Libvirt**(*name*, *host*, *port*, *master*, *run=True*)
Bases: vespa.agent.Agent

connect_link(*nodeName='arch-poc-win'*)

contains_vm(*vm*)

cut_link(*nodeName='arch-poc-win'*)

launch()

migrate(*nodeName*, *quarantine*, *quarantine_user*)

restart(*vm*)

restart_hard(*vm*)

send(*msg*)

send_key(*vm*, *args*)

1.11 vespa.controller module

Controller

class vespa.controller.**Controller**(*model*, *view*, *testmode=False*)
Bases: object

handler(*signum*, *false*)

start()

class vespa.controller.**HttpServer**(*name*, *host*, *port*, *handler*, *c*)

start()

stop()

class vespa.controller.**HttpServerHandler**(*request*, *client_address*, *server*)
Bases: BaseHTTPServer.BaseHTTPRequestHandler

do_GET()

log_message(*format*, **args*)

class vespa.controller.**MyHTTPServer**(*server_address*, *RequestHandlerClass*, *handler*, *control*)
Bases: BaseHTTPServer.HTTPServer

this class is necessary to allow passing custom request handler into the RequestHandlerClass

vespa.controller.**server_handler**(*c*, *request*)

1.12 vespa.ho module

Horizontal orchestrator

class vespa.ho.**HO**(*name*, *host*, *port*, *master*, *run=True*)
Bases: vespa.node.Node

findAgent(*name*)

1.13 vespa.ho_hy module

Horizontal orchestrator

class vespa.ho_hy.**HO_HY**(*name*, *host*, *port*, *master*, *run=True*)
Bases: vespa.ho.HO

Create an horizontal orchestrator to handle agents at the hypervisor level.

Returns The HO to gather and react on hypervisor agents.

Return type Node

ninjaMethod()
Empty function for tests

send(*msg*)
Overload the internal send() to capture and send messages to the backend

Parameters **msg** (*str*) – The massage to process and to send

Returns The backend response

Return type str

1.14 vespa.ho_ph module

Horizontal orchestrator

class vespa.ho_ph.**HO_PH**(*name*, *host*, *port*, *master*, *run=True*)
Bases: vespa.ho.HO

Create an horizontal orchestrator to handle agents at the physical level.

Returns The HO to gather and react on physical agents.

Return type Node

ninjaMethod()
Empty function for tests

send(*msg*)
Overload the internal send() to capture and send messages to the backend

Parameters **msg** (*str*) – The massage to process and to send

Returns The backend response

Return type str

1.15 vespa.ho_vm module

Horizontal orchestrator

class vespa.ho_vm.**HO_VM**(*name*, *host*, *port*, *master*, *run=True*)
Bases: vespa.ho.HO

Create an horizontal orchestrator to handle agents at the VM level.

Returns The Horizontal Orchestrator to gather and react on VM agents.

Return type Node

ninjaMethod()
Empty function for tests

send(*msg*)
Overload the internal send() to capture and send messages to the backend

Parameters **msg** (*str*) – The massage to process and to send

Returns The backend response

Return type str

1.16 vespa.log_pipe module

log_pipe

class vespa.log_pipe.**bcolors**
Bases: object

ENDC = '\x1b[0m'

FAIL = '\x1b[91m'

HEADER = '\x1b[95m'

OKBLUE = '\x1b[94m'

OKGREEN = '\x1b[92m'

WARNING = '\x1b[93m'

disable()

vespa.log_pipe.**debug1**(*str*)

vespa.log_pipe.**debug2**(*str*)

vespa.log_pipe.**debug4**(*str*)

vespa.log_pipe.**debug5**(*str*)

vespa.log_pipe.**debug_comm**(*str*)

vespa.log_pipe.**debug_comm_detail**(*str*)
Display extended communications information - How sendRemote split RECV_LENGTH

Notes: offloaded to prevent screen flooding

vespa.log_pipe.**debug_comm_len**(*s*)
Display maximum sized informations

Notes: offloaded to prevent screen flooding

vespa.log_pipe.**debug_controller**(*str*)

vespa.log_pipe.**debug_crypto**(*str*)

vespa.log_pipe.**debug_info**(*str*)

vespa.log_pipe.**debug_init**(*str*)

vespa.log_pipe.**debug_thread**(*str*)

1.17 vespa.model module

Model

class vespa.model.**Model**
Bases: vespa.node.Node

create_object_instance(*config*, *obj*, *master*)

findNode(*name*)
Return a tuple if the node "name" is found, raise an Exception otherwise. TODO: Refactor (3x)

find_vo(*config*)
Return VO object from config file

One and only one VO

sendRemoteWake(*remote*, *msg*)
Force sending content to a remote host. Loop until it is done

1.18 vespa.node module

Most basic inherited class for a simple Node

Default config without backend

Pthread is used as Profiler wrapper

If you do not need profiling you may replace the PThread class with: class Node(Thread):

INTERNALS:

class vespa.node.**Node**(*name*, *host*, *port*, *master*, *run=True*)
Bases: vespa.node.PThread

run()

class vespa.node.**PThread**(*name*, *host*, *port*, *master*, *run=True*)
Bases: threading.Thread

desc()
Return the tuple representing a node

destroy()
Destroy all slaves

findNode(*name*)

get_backend()
Return the backend registered on initialization (i.e. Resource)

launch()

list_slaves()
Return self.slaves

listen_interface(*host*)

register(*name*, *host*, *port*)

register_alert_handler(*handler*)

run()
Thread listenning on node port. It creates a worker thread for each accepted socket.

It SHOULD NOT accept multiple hosts, but ready for it.

send(*msg*)
Provide an entry to current node functions.

sendAlert(*msg*)
Wrapper for sendRemote with alert formatting.

See "sendRemote" for arguments description and returns

sendRemote(*remote*, *msg*, *needack=True*)
Send a message to a node (remote) using the node.desc() string. This function deals with sockets directly.

Default behavior is to wait data as acknowledgement (needack). It is only modified for messages needing fast delivery and processing such as alerts.

sendRemotef(*remote*, *msg*)
Wrapper for sendRemote with needack=False

See "sendRemote" for arguments description and returns

sendSocket(*s*, *remote*, *msg*)
Handle the socket (s.send) message and encryption routines.

wait_backend(*max_tries=0*)
Ping node backend and return when backend is up.

WARNING: Does not timeout if max_tries = 0

worker(*conn*)
Handle socket reception job.

class vespa.node.**ThreadWorker**(*group=None*, *target=None*, *name=None*, *args=()*, *kwargs=None*, *verbose=None*)
Bases: threading.Thread

run()

1.19 vespa.starter module

Starter

1.20 vespa.view module

class vespa.view.**View**(*model*)
Bases: object

1.21 vespa.vo module

Vertical orchestrator

class vespa.vo.**VO**(*name*, *host*, *port*, *master*, *run=True*)
Bases: vespa.node.Node

Create a Vertical Orchestrator to interconnect all other components. It may be requested by an external controller. All incoming communications go through the *alert()* method, and are parsed there.

Returns The VO instance

Return type Node

alert(*msg*)
This is the most important function of the whole framework. The format is *alert|source>...>source>message*. The message is also split in the following format *function#arg1#...#argN*. The two formats are used to create a Finite State Machine, each alert being a state transition.

Parameters **msg** (*str*) – The massage to process with the current format

get_alerts()

get_ip_connections()

get_link_stats()

get_next_recv_bytes()

get_next_trans_bytes()

get_recv_bytes()

get_topology()

get_trans_bytes()

1.22 Module contents

CHAPTER 2

Indices and tables

- *genindex*
- *modindex*
- *search*

V

A

B

C

D

E

F

G

W