

Technical Support DATA (EMEA)

System Functional and Design Specification

Preventive Maintenance

Author(s): Raphael Voyer, Jonathan Trebaol

Revision: 0.1

Date: 29/07/2021

Copyright © 1995-2016 Alcatel-Lucent, Incorporated

ALL RIGHTS RESERVED WORLDWIDE

Alcatel-Lucent DSD

26801 West Agoura Road, Calabasas, CA 91301

(818) 880-3500

TABLE OF CONTENTS

[1 INTRODUCTION 4](#_Toc78494220)

[1.1 Purpose 4](#_Toc78494221)

[1.2 Intended Audience 4](#_Toc78494222)

[2 FUNCTIONAL DESCRIPTION 5](#_Toc78494223)

[2.1 Basic Overview 5](#_Toc78494224)

[2.2 Platforms Supported 8](#_Toc78494225)

[2.3 Prerequisites 8](#_Toc78494226)

[3 Set-up the environment 9](#_Toc78494227)

[4 DEBIAN SERVER INITIALIZATION 10](#_Toc78494228)

[4.1 Introduction 10](#_Toc78494229)

[4.2 Services 10](#_Toc78494230)

[4.2.1 TFTP 10](#_Toc78494231)

[4.2.2 Rsyslog 10](#_Toc78494232)

[4.2.3 Logrotate 12](#_Toc78494233)

[4.2.4 Iptables 13](#_Toc78494234)

[4.2.5 Python Utilities 13](#_Toc78494235)

[5 PYTHON Utils SCRIPTS 14](#_Toc78494236)

[5.1 Support Tools Script 14](#_Toc78494237)

[5.2 Activate the Syslog messages 16](#_Toc78494238)

[5.3 Send Notification 16](#_Toc78494239)

[5.4 Web Receiver 16](#_Toc78494240)

[5.5 Requests Handler 16](#_Toc78494241)

[6 USE CASE PYTHON SCRIPTS 17](#_Toc78494242)

[6.1 Log Debug debug2 17](#_Toc78494243)

[6.2 Log Debug debug3 18](#_Toc78494244)

[6.3 L2 Loop Script 18](#_Toc78494245)

[6.4 Port Flapping Script 20](#_Toc78494246)

[6.5 Port Scanning (DDOS) Script 22](#_Toc78494247)

[6.6 Collection Logs (Switch) Script 23](#_Toc78494248)

[6.7 Collection Logs (Stellar AP) Script 23](#_Toc78494249)

REVISION HISTORY

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Revision | Date | Agile Revision | Update By | Revision Description | Status |
| 0.1 | 29/07/2021 |  | Raphael Voyer | Document creation | Draft |

# INTRODUCTION

## Purpose

This document describes the Software Requirements Specification and Functional Specification for the project “Preventive Maintenance”

Preventive maintenance allows the automation of troubleshooting on OmniSwitch AOS 8.x, Stellar APs equipment, OV 2500 and add a notification function either by email or by Rainbow. It consists of a Debian server that will receive logs of the various devices from the network. All logs are received and processed by Rsyslog. Rsyslog will both detect one or more patterns within the logs, and then stores them and runs a script based on the received log. The scripts are developed in Python.

## Intended Audience

This document is intended for the following organization:

* Support organization providing end-user support.

# FUNCTIONAL DESCRIPTION

## Basic Overview

Here an example of loop detection and resolution:

1. Server receives a syslog message containing the pattern Buffer list is empty from OmniSwitch

2. This pattern matches with a Rsyslog rule and a script is executed in order to increase the debug level 3. Server receives of a syslog message containing the pattern slnhwlrncbkhandler AND port AND bcmd’

4. This pattern matches with a Rsyslog rule and a script is executed:

a. The script detects a loop by parsing the logs and gets the interface port number

b. The script sends a notification to the Administrator via Rainbow and/or email

5. If the Administrator answers "Yes" to the request, script disables the interface to resolve the loop issue

6. The script sends a notification to Administrator once issue is resolved

To be able to use the Preventive Maintenance, the folder must contain the following files (detailed in section [Set-up the environment](#_Toc76555060)):  
[Setup.sh](#_DEBIAN_SERVER_INITIALIZATION): Allows configuration of the Debian server.  
Devices.csv: IP addresses’ list of the switches to push the command swlog output socket <server>  
[support\_active\_output\_socket.py:](#_Active_Output_Socket) Script to push the command swlog output socket to switches listed into Devices.csv  
[support\_tools.py](#_Support_Tools_Script): This script contains the common functions for preventive maintenance like extract an IP Address, connect to switch by SFTP

[support\_send\_notification.py](#_Send_Notification): This script contains the functions for sending email or Rainbow notifications  
[support\_response\_handler.py](#_Requests_Handler): This script is handling the notification requests (turn on web server for mail request, send request by mail and by rainbows)  
[support\_web\_receiver\_class.py](#_Web_Receiver): This script loads a web server on the fly for receiving answers from Administrator following email notification

In this first release of Preventive Maintenance, the environment is initialized by executing the bash script **Setup.sh**

Administrator is prompted to provide information that will help set up the components

Following operations are done within this script:

* Select the notification engine (Rainbow/Email)
* Provide the Patterns for collecting devices logs
* Provide the Switch’s credentials
* Provide the Stellar AP’s credentials
* Subnets of devices which we are collecting/processing logs
* Installation of Python3 and modules
* Rsyslog configuration
* Logrotate configuration
* IPTables configuration (to be implemented)
* TFTP Server installation and configuration
* Activation of socket output on Switches
* Create /opt/ALE\_Script working directory

All Administrator information is collected and save into /opt/ALE\_Script/ALE\_script.conf.

Example of ALE\_script.conf content:

admin,switch,email1@al-enterprise.com;email2@al-enterprise.com,59fe823c15xxxxxxxxxxxx4feef59a@openrainbow.com,sender@gmail.com,pass\_mail\_sender,10.130.7.14,support,Letacla01\*,Letacla01\*,6573953192,,,

This structure file is a list where each data is separated by ‘,’:

* Switches Login
* Switches Password
* Admin emails( if more than 1 email separate each by ‘;’)
* Rainbow JID
* Email used for sending logs and notifications
* Ip address of the Debian Server.
* If AP are used:
  + AP login
  + AP password
  + AP technical support code
* Client ID, which is a random number between 0000000000 and 9999999999 created at the execution of Setup.sh. (more details in section [Web Receiver](#_Web_Receiver))

What is the JID:

The JID is a unique ID referring to an user or a bot on Rainbow.

How to obtain the JID:

Search for Rainbow bubble “who am I” and say “hello”:



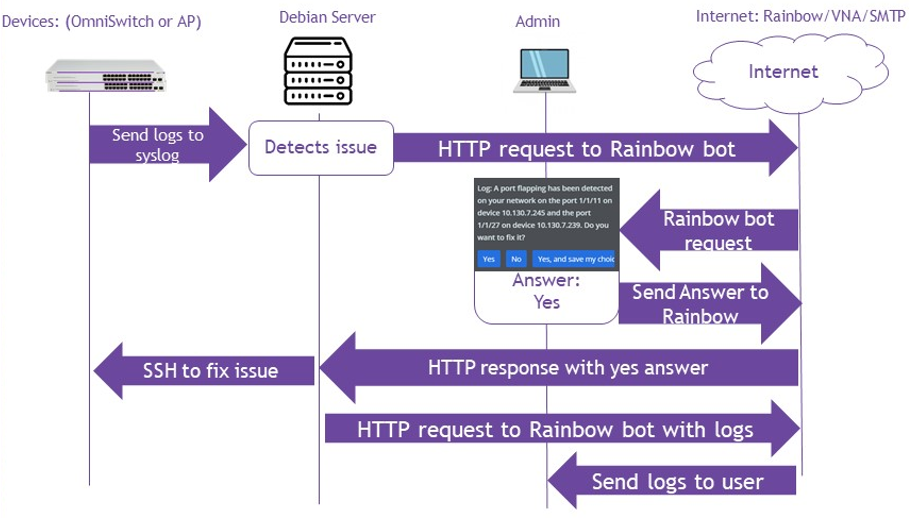
The bot will answer with your JID and ID

How to use the JID?

We support up to 3 JIDs with the current workflow by calling the REST-API as following:

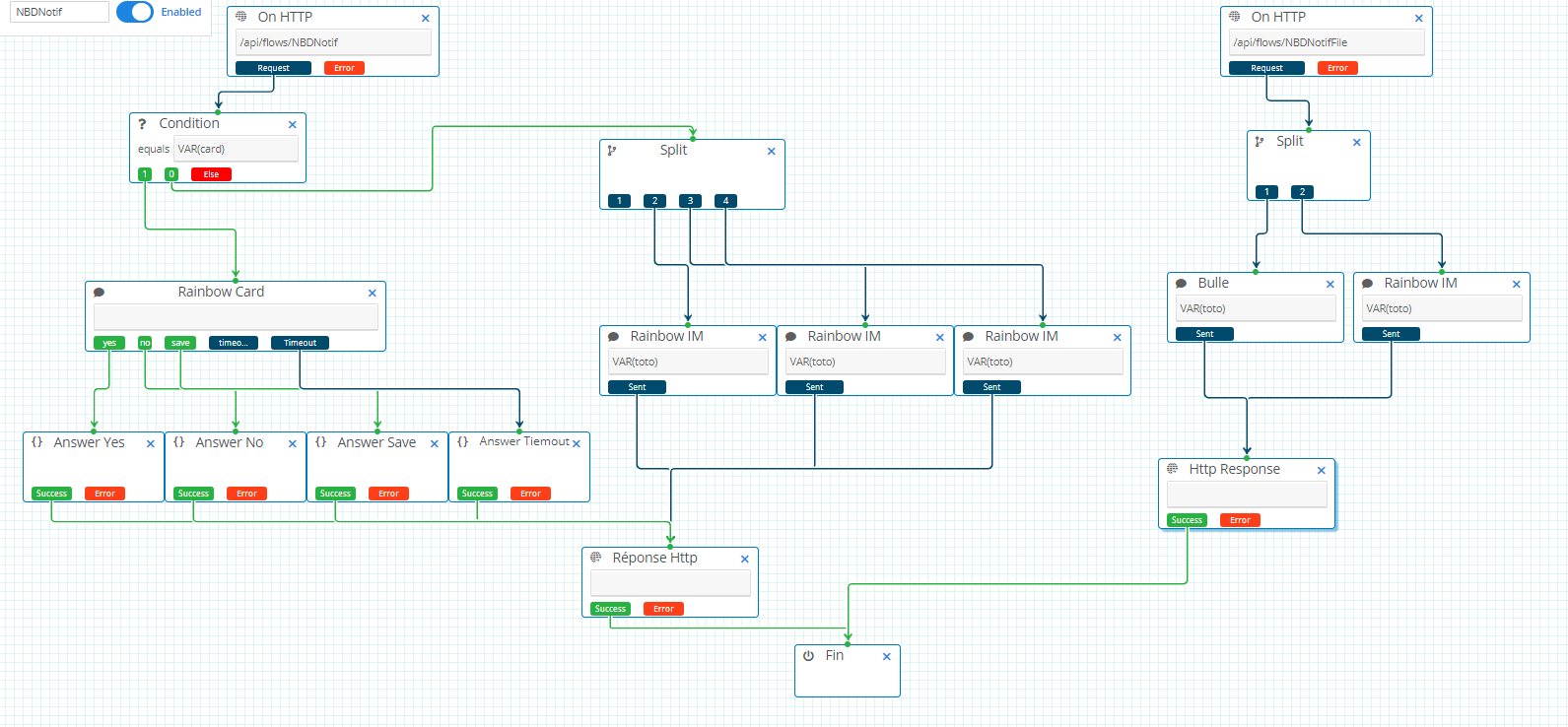
<https://tpe-vna.al-mydemo.com/api/flows/NBDNotif/?toto=tata&jid1=xxxxx&jid2=xxxxxx&jid3=xxxx>

Call flow when Rainbow bot is used for notifications:



What is the purpose of VNA:

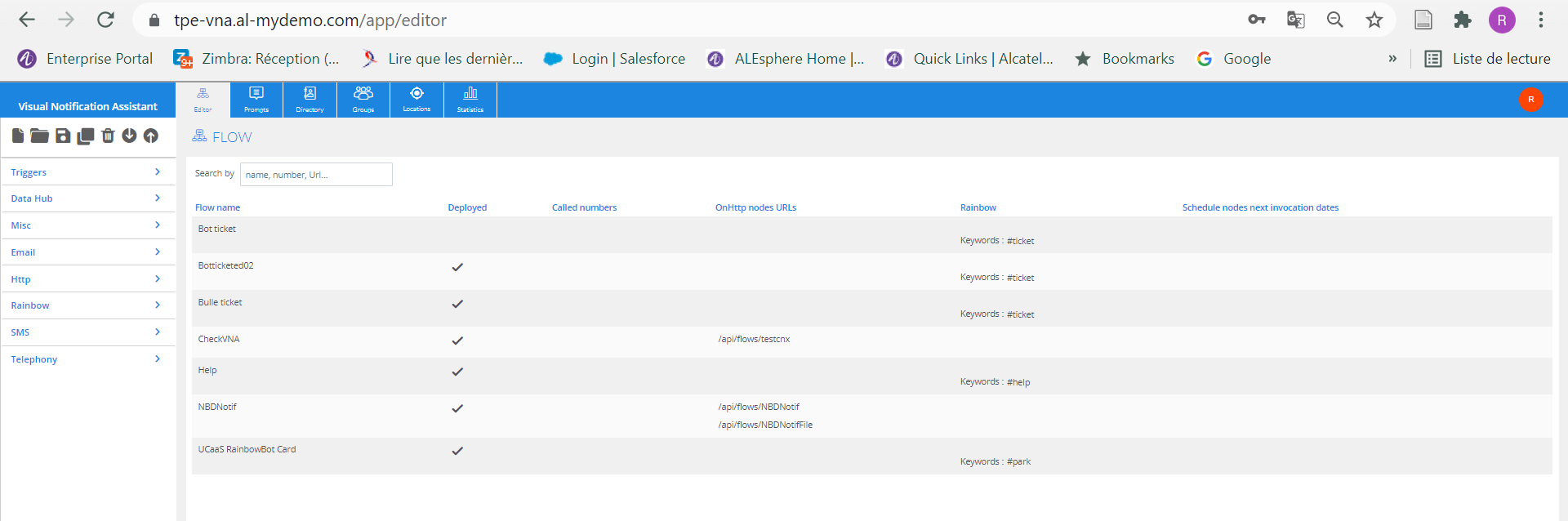
VNA for [**V**isual **N**otification **A**ssistant](https://www.al-enterprise.com/fr-fr/produits/applications/visual-notification-assistant)  is a workflow engine used for doing interaction between the Server and Rainbow. Based on inputs, workflow is generating notifications, notifications cards, sending logs to Rainbow bot/JID. Below the current workflow created for the project needs:



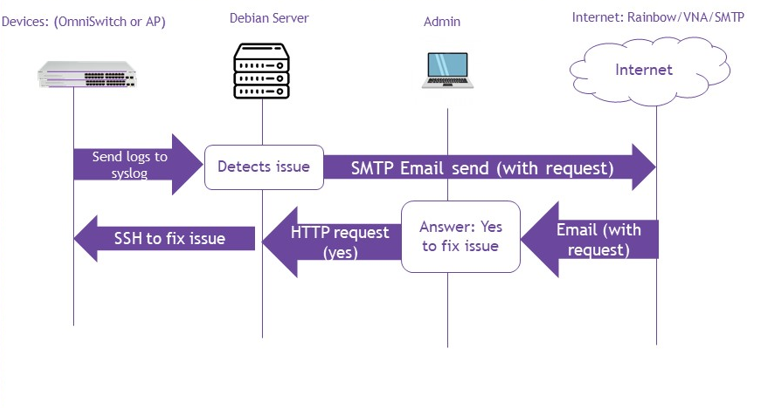
To access and configure the VNA workflow, go to this URL : <https://tpe-vna.al-mydemo.com/app/login>

Contact: PEYREBESSE Thierry [thierry.peyrebesse@al-enterprise.com](mailto:thierry.peyrebesse@al-enterprise.com) ALE PreSales

Then, click on editor and select the workflow “NBDNotif”:



Call flow when Email is used for notifications:



**Note**: Here in the call flow we use GMAIL as SMTP Relay Service (SSL, port 465) to deliver our messages

## Platforms Supported

Preventive Maintenance is supported on OmniSwitches running AOS 8.x and AP Stellar 4.0.x and above (all models), OV 2500 4.5R02 and above.

## Prerequisites

Following components were used:

Supported OS: Debian, Raspbian

Debian Server: 9.13

SSH: OpenSSH\_7.4p1

Python: Python 3.5.3 and above

SMTP Relay Service: smtp.gmail.com, SSL, port 465

# Set-up the environment

How to set-up the Preventive Maintenance Environment:

**Step1**:Download the Git folder <https://github.com/data-support-ale/Script_Preventive_Maintenance.git>

**Step2**: cd /Script\_ALE

**Step3**: sudo ./Setup.sh then follow configuration steps

Check that working directory /opt/ALE\_Script is created with following files:

ALE\_script.conf

support\_AP\_get\_log.py

support\_response\_handler.py

support\_send\_notification.py

support\_switch\_debugging\_ddos.py

support\_switch\_debugging.py

support\_switch\_enable\_qos.py

support\_switch\_get\_log.py

support\_switch\_pmd.py

support\_switch\_port\_disable.py

support\_switch\_port\_flapping.py

support\_tools.py

support\_update\_iptables\_OV2500.py

support\_web\_receiver\_class.py

support\_active\_output\_socket.py

support\_web\_receiver.py

Devices.csv

Setup.sh script will save the environment information and preventive actions into the following files:

device\_catalog.conf IP addresses extracted from Device.csv

decisions\_save.conf List the decisions collected from notifications (rainbow card, email) for a particular case, example if Administrator asked for fixing permanently the loop detected on switch 192.168.80.27 port 1/1/4 the following entry is created: 192.168.80.27,1/1/4,loop,always

configqos the QOS Policies pushed on the OmniSwitch when an Attacker is blocked

# DEBIAN SERVER INITIALIZATION

## Introduction

The first step is to execute the bash script "Setup.sh" that will set-up the environment (create directories/sub-directories, install and configure components as described on previous [section](#_Set-up_the_environment)). The purpose of the Setup.sh script is to configure all the services that are needed for Preventive Maintenance scripts running.

## Services

All services described below are fully installed and configured by the Setup.sh script. To prevent modifying the Rsyslog configuration file, Administrator can re-execute the script Setup.sh in order to modify the rules “Patterns” or the network environment.

### TFTP

A TFTP Server is installed on the server, the default directory is /tftpboot/

This service is required for uploading logs from Stellar APs.

TFTP Server is used by the following function:

* Collect the AP Stellar logs (we execute remotely the script take\_snapshot.sh start)

Commands to verify the configuration:

systemctl status tftpd-hpa (the service must be in running mode)

admin-support@debian2:/opt/ALE\_Script$ systemctl status tftpd-hpa

● tftpd-hpa.service - LSB: HPA's tftp server

Loaded: loaded (/etc/init.d/tftpd-hpa; generated; vendor preset: enabled)

Active: active (**running**) since Fri 2021-07-23 21:56:14 CEST; 5 days ago

cat /etc/default/tftpd-hpa:

# /etc/default/tftpd-hpa

TFTP\_USERNAME="tftp"

TFTP\_DIRECTORY="/tftpboot"

TFTP\_ADDRESS="0.0.0.0:69"

TFTP\_OPTIONS="-l -c -s"

### Rsyslog

Rsyslog Server receives syslog messages from devices (OmniSwitches, Stellar APs) and based on patterns execute the following tasks:

- convert the syslog message into json format and stored in path /var/log/devices/lastlog\_xx.json

- forward the logs to specific files for further processing in path: /var/log/devices/<devices\_hostname>/syslog.log

- execute python scripts /opt/ALE\_Scripts/support\_<device\_type>\_<use\_case>.py

Commands to verify the configuration:

systemctl status rsyslog (the service must be in running mode)

admin-support@debian2:/opt/ALE\_Script$ sudo systemctl status rsyslog

● rsyslog.service - System Logging Service

Loaded: loaded (/lib/systemd/system/rsyslog.service; enabled; vendor preset: enabled)

Active: active (**running**) since Wed 2021-07-28 14:29:43 CEST; 1 day 7h ago

A temporary file is created for each supported use case in order to perform pre-processing:

lastlog.json

lastlog\_ddos.json

lastlog\_ddos\_ip.json

lastlog\_flapping.json

lastlog\_loop.json

Here the definition of Rsyslog rules and actions executed:

Diagram

Description automatically generated

Extract of the /etc/rsyslog.conf with rules for executing loop detection and resolution:

* Only the subnets set on the $AllowedSender are processed, all others messages are discarded
* The modules omprog is loaded for executing the python scripts
* Rsyslog is listening on UDP Port 514
* Templates are defined to convert the syslog messages into json format
* Rules are defined to detect specific pattern when a loop occurs, messages containing these pattern are forwarded into json format for processing and python script is executed for further actions (enable debug logs, disable the interface port)
* Last rule is used to log all Rsyslog script executions

# /etc/rsyslog.conf Configuration file for rsyslog.

#

# For more information see

# /usr/share/doc/rsyslog-doc/html/rsyslog\_conf.html

### Allowed IP Addresses ####

$AllowedSender UDP,127.0.0.1, 10.130.7.0/24, 10.69.145.0/24, 192.168.80.0/24

#################

#### MODULES ####

#################

module(load="imuxsock") # provides support for local system logging

module(load="imklog") # provides kernel logging support

module(load="omprog") # provides support for script

# provides UDP syslog reception

module(load="imudp")

input(type="imudp" port="514")

### Template definition ####

$template DynamicFile,"/var/log/devices/%hostname%/syslog.log"

template (name="devicelog" type="string"

string="/var/log/devices/lastlog.json")

template (name="deviceloghistory" type="string"

string="/var/log/devices/%fromhost-ip%\_%$YEAR%-%$MONTH%-%$DAY%\_history.json")

template(name="json\_syslog"

type="list") {

constant(value="{")

constant(value="\"@timestamp\":\"") property(name="timereported" dateFormat="rfc3339")

constant(value="\",\"type\":\"syslog\_json")

constant(value="\",\"relayip\":\"") property(name="fromhost-ip")

constant(value="\",\"hostname\":\"") property(name="hostname" caseconversion="lower")

constant(value="\",\"message\":\"") property(name="rawmsg" format="json")

constant(value="\",\"end\_msg\":\"")

constant(value="\"}\n")

}

\*.\* ?DynamicFile;json\_syslog

# Rules

#

if $msg contains 'Buffer list is empty' then {

$RepeatedMsgReduction on

action(type="omfile" DynaFile="deviceloghistory" template="json\_syslog" DirCreateMode="0755" FileCreateMode="0755")

action(type="omfile" DynaFile="devicelog" template="json\_syslog" DirCreateMode="0755" FileCreateMode="0755")

queue.type="LinkedList" queue.size="1" queue.workerThreads="1")

stop

}

if $msg contains 'slnHwlrnCbkHandler' and $msg contains 'port' and $msg contains 'bcmd' then {

$RepeatedMsgReduction on

action(type="omfile" DynaFile="deviceloghistory" template="json\_syslog" DirCreateMode="0755" FileCreateMode="0755")

action(type="omfile" DynaFile="devicelogloop" template="json\_syslog" DirCreateMode="0755" FileCreateMode="0755")

action(type="omprog" binary="/opt/ALE\_Script/support\_switch\_port\_disable.py" queue.type="LinkedList" queue.size="1" queue.workerThreads="1")

stop

}

:syslogtag, contains, "montag" /var/log/devices/script\_execution.log

& stop

### Logrotate

Logrotate limits the size of log files stored in/var/log.

For each log file, Logrotate performs 2 simultaneous operations:

• Rotation: It archives the log file under a different name and deletes the oldest archive, a SIGHUP signal instead of a service restart is sent to Rsyslog to avoid packet loss during rotation

• compression: it may compress the log file before archiving it

Logrotate is configured to work on logs located on /var/log/devices/ files with the extensions .json or .log, and compress them Daily into .gz format.

Configuration file is located at path: /etc/logrotate.d/Rsyslog

For testing the log rotation, execute the command: logrotate -d /etc/logrotate.d/rsyslog

### Iptables

To be implemented

### Python Utilities

Following packages are installed during Setup.sh execution:

Python3: Package to develop and execute Python scripts.

Sshpass: Send commands by SSH without human interaction, the password is directly entered in the command.

Pysftp: Used for the SFTP file transfer with Python code

Flask: Used for the creation and configuration of a WebServer with Python code.

Tftpd-hpa: Used for the creation and configuration of a TFTP server on the Debian server.

# PYTHON Utils SCRIPTS

## Support Tools Script

Name: support\_tools.py

Contains the functions to perform remote action on OmniSwitches and detect incidents.

This script must be updated when a new signature/use case is supported

Functions:

enable\_debugging(user,password,ipadd):

Send the command swlog appid bcmd subapp 3 level debug2 to the switch in arguments

disable\_debugging(user,password,ipadd):

Send the command swlog appid bcmd subapp all level info to the switch in arguments

enable\_debugging\_ddos(user,password,ipadd):

This function enables the debugging level 3 on the switch put in arguments to get more details in log for processing

disable\_debugging\_ddos(user,password,ipadd):

This function disables the debug level 3 on the switch in argument back to info.

disable\_port(user,password,ipadd,portnumber):

Send the command interfaces port admin-state disable to the switch and port in arguments .

enable\_port(user,password,ipadd,portnumber):

This function enables the port where there is a loop on the switch put in arguments.

enable\_qos\_ddos(user,password,ipadd,ipadd\_ddos):

Use file\_setup\_qos(ipadd\_ddos) then push the file configqos and apply the configuration on switch in arguments.

disable\_qos\_ddos (user,password,ipadd,ipadd\_ddos):

Use file\_unset\_qos(ipadd\_ddos) then push the file configqos and apply the configuration on switch in arguments.

file\_setup\_qos(ipadd):

Create QOS Policies to block the Attacker

file\_unset\_qos(ipadd):

Create the QOS Policies to unblock the Attacker

send\_python\_file\_sftp(user,password,ipadd,filename):

Send file in the /flash/python from /opt/ALE\_Script/ Debian server folder.

get\_file\_sftp(user,password,ipadd,filename):

Get a file from switch and put in the /tftpboot/ Debian server directory.

detect\_port\_loop():

This function detects if there is a loop in the network (support more than 10 logs in 2 seconds)

detect\_port\_flapping():

This function detects if there is port flapping in the logs

If there is more than 5 logs within 10 seconds, we consider there is a port flapping.(10 seconds is set for the demonstration)

save\_attachment(ipadd):

Copy the 30th last logs of the equipment in parameter in the /var/log/devices/attachment.log file. This log is then attached in the email or Rainbow JID

save\_attachment\_deauth (ipadd,device\_mac,timestamp):

extract\_ip\_port():

This function collects the IP address of the switch from syslog messages.

if, this is a loop use case, the function collects also the port number, otherwise the port is set as 0.

extract\_ip\_ov():

This function extracts the IP address of all devices in the device catalog (Devices.cvs).

extract\_ip\_ap():

This function extracts the IP address of all devices in the device catalog.

extract\_ip\_ddos():

This function checks if there is a ddos attack.

check\_timestamp():

This function provides the time between the last log and the current log.

get\_credentials():

This function collects the credentials from OmniSwitches and credentials of the SMTP Relay Service

replace\_logtemp():

This function put the last log in logtemp file to verify the timestamps between two L2 loops

get\_credentials\_ap():

This function collects the AP's credentials in the file ALE\_script.conf.

get\_mail():

This function collects Mail information in the file ALE\_script.conf.

get\_id\_client():

get\_server\_log\_ip():

This function collects the IP Address of log server in the file ALE\_script.conf.

get\_jid():

This function collects Rainbow JID in the file ALE\_script.conf.

add\_new\_save(ipadd,port,type,choice = "never"):

This function saves the instruction received from Rainbow card.

## Activate the Syslog messages

Name: support\_active\_output\_socket.py

This script is executed when setting-up the environment by Setup.sh. Based on the IP Addresses listed into Devices.xls, this script will add the following command swlog output socket <server\_ipaddress> required for sending syslog messages

## Send Notification

Name: support\_send\_notification.py

Contains functions for sending email and rainbow notifications.

Functions:

**Send\_message**(info,jid):

This function sends a url request to the bot, to send a rainbow message to the Admin

**send\_message\_request**(info,jid,receiver):

This function sends a url request to the bot, to sends a Rainbow card (request) to the Admin

**send\_file**(info,jid,ipadd,filename =''):

This function sends a url request to the bot, to sends a file by Rainbow to the Admin

**send\_mail\_request**(ip\_switch\_1, ip\_switch\_2,info,subject,gmail\_user,gmail\_password,mails,ip\_server,id\_client,id\_case):

This function sends a mail to the Admin, with the request (yes, no, yes and save my decision)

**send\_mail**(ip\_switch\_1,ip\_switch\_2,info,subject,gmail\_user,gmail\_password,mails,timestamp = 0 ,mac = "0" ):

This function sends a mail to the Admin.

## Web Receiver

Name: support\_web\_receiver\_class.py

Start a WebServer application with keys and values in URL parameters. This function is used to collect answers from Administrator when using email notification

For example:

When the WebServer is started, 3 pages are available (yes, no, and save):

Id client: 0123456789

Id case: 2464321088

Yes:<http://10.130.7.14:5200?id=01234567892464321088&answer=yes>  
no:<http://10.130.7.14:5200?id=01234567892464321088&answer=no>  
save: <http://10.130.7.14:5200?id=01234567892464321088&answer=save>

For security concern, the other parameters to access the correct page are the Id client and Id case.

When Administrator clicks on the URL, answer is received by server and immediately the WebServer is stopped.

If there is no URL access from Administrator after 60 seconds, we consider the answer Yes.

This script is called by the request handler

## Requests Handler

Name: support\_response\_handler.py

When an issue occurred, a notification was generated by mail and/or by Rainbow to the Administrator. To ensure the answer given by the Administrator is not matching with another issue occurring at the same time we use the client ID created during the Setup.sh execution and a Case ID creates at the execution of the support\_response\_handler.py script.

Then, we execute 2 services at the same time (threading method) for the same use case for both notifications send request email and send rainbow request.

The script starts the Web service, with the correct parameters for the case (with ID numbers)

The script sends the email with the correct ID numbers in URL

The script sends the request by rainbow.

If the answer is received by the Web service, the Web service is turn off and the answer given is processed

If the answer is received by the Rainbow card, we send the command to the Web service to turn it off and the answer given is processed

If no answer given for 60 seconds, we send the command to the Web service to turn it off and the answer “yes” is processed

The script uses the content file save\_decision.conf which list all answers received from the Administrator

The save\_decision.conf structure file:

10.130.7.247,1/1/33,flapping,never

192.168.80.27,0,ddos,always

192.168.80.27,1/1/19,flapping,always

192.168.80.25,1/1/19,flapping,always

192.168.80.27,1/1/4,loop,always

192.168.80.27,1/1/35,loop,never

* Check if the case is already saved in the file:
  + Always, the answer will be automatically return as yes in the main script
  + Never, the answer will be automatically return as no in the main script
* If not, the script continues
* The script gets the id of the client, and create an id for the case, this is a random number between 0000000000 and 9999999999.
* Creation of the WebServer (Web Receiver)
* Send request by email
* Send request by Rainbow
* Wait for turning off the Web service
* Then return the answer to the main script and save the decision if necessary.

Request\_handler is used every time the main script is sending a notification to Administrator.

# USE CASE PYTHON SCRIPTS

The pattern in the logs received from devices are not enough for processing and apply a resolution.

For this purpose, functions are defined for enabling debug logs when specific patterns are detected in the logs. In case the same pattern is received several times, we consider the frequency of same messages, the quantity of received messages and Timestamps in the processing.

The scripts execute commands on OmniSwitches for enabling debug logs.

## Log Debug debug2

Name: support\_switch\_debugging.py

Rsyslog Trigger: ‘Buffer list is empty’

Last log file: lastlog.json

Log example:

{"@timestamp":"2021-06-07T10:03:49+02:00","type":"syslog\_json","relayip":"10.130.7.251","hostname":"os6860e-core1","message":"<131>Jun 7 10:03:49 OS6860E-Core1 swlogd bcmd rpcs ERR: rpcUtilBufListGet:155 Buffer list is empty","end\_msg":""}

The purpose of this script is to enable debugging on the appid “bcmd”. Indeed, if the debug mode is set on ‘info’(default value) all logs received by Rsyslog will contain the same message “Buffer list is empty”. To obtain detailed log with interface port number where loop is detected, we need to enable the logs in debug2.

We use the enable\_debugging function from support\_tools which establishes a SSH session with the use of module sshpass to access remotely on the OmniSwitch and apply the following command:

swlog appid bcmd subapp 3 level debug2

Once the switch’s logs are set in debug mode, the script clears the log file lastlog.json.

## Log Debug debug3

Name: support\_switch\_debugging\_ddos.py

Rsyslog Trigger: ‘Denial of Service attack detected: <port-scan>’

Last log file: lastlog\_ddos.json

Same method as previous section, when the script is executed, the following CLI command is applied remotely thru SSH session swlog appid ipv4 subapp all level debug3

## L2 Loop Script

Name: support\_switch\_port\_disable.py

Rsyslog Trigger: ‘slnHwlrnCbkHandler' ;'port';'bcmd'

Last log file: lastlog\_loop.json

Log example:

{"@timestamp":"2021-05-26T11:41:25+02:00","type":"syslog\_json","relayip":"192.168.80.27","hostname":"os6860e-2-gartner","message":"<135>May 26 11:41:25 OS6860E-2-Gartner swlogd bcmd rpcs DBG2: slnHwlrnCbkHandler:648 port 19 mod 0 auth 0 group 0","end\_msg":""}

This scripts detects a L2 loop in the network by analyzing the log file lastlog\_loop.json, shutdown the port administratively which is responsible of the loop and set back the switch in debug mode “Info”.

Step1: gets all information to connect to the switch and notify by Rainbow and/or email.

Step2: extracts the switch IP and the interface port number responsible of the loop

**Condition one:** no loop has been detected for 10 seconds.

The script uses check\_timestamp() function. In this function, we use 2 files, lastlog\_loop and a new file logtemp. In this new file, we will register the lastlog that we use to stop the previous loop.

* If logtemp doesn’t exist or there is not only 1 line in logtemp we copy the first lastlog\_loop log.
* We extract the timestamps from the first line of lastlog\_loop and the timestamp from logtemp.
* We return the difference of the two timestamps then we replace the log in logtemp by the new log.

**Condition two:** 10 loop logs received in less than 2 seconds.

* Check if there is more than 10 lines in lastlog\_loop. Else we return 0
* Extract the timestamps of the first line and the 10th line of lastlog\_loop
* Change time to decimal to get a continuous number
* If there is less than 2 seconds between the two timestamps, we return 1 else we return 0
* Send a request by mail and by Rainbow card to fix or not the issue:
  + If the answer is **yes**, the script sends thru SSH a command to disable the port, and enable the port(s).
  + If the answer is **yes and save my decision** sends thru SSH a command to disable the port(s). Then save the answer in the decisions\_save.conf file
  + If the answer is **no**, the script stops, and saves the decision in the decisions\_save.conf file.
* Send the log file and a message by Rainbow
* Send the log in attachment and an email to the email address stored in ALE\_script.conf
* Then we disable the debugging on the switch.

If the condition one is not respected the script clear the file last log to get in first position a most recent log.

If the condition 2 is not respected the script does nothing.

Flowchart:

Diagram

Description automatically generated

## Port Flapping Script

Name: support\_switch\_port\_flapping.py

RsyslogTrigger: 'pmnHALLinkStatusCallback:206'

Last log file: lastlog\_flapping.json

Log example:

{"@timestamp":"2021-07-08T10:46:33+02:00","type":"syslog\_json","relayip":"192.168.80.27","hostname":"os6860e-2-gartner","message":"<134>Jul 8 10:46:33 OS6860E-2-Gartner swlogd portMgrNi main INFO: : [pmnHALLinkStatusCallback:206] LINKSTS 1\/1\/3 DOWN (gport 0x2) Speed 0 Duplex HALF","end\_msg":""}

This script is to detect a port flapping in a network by analyzed the log file lastlog-flapping.json, restart the port which responsible of the flapping and return back the switch in debug mode info.

Script steps:

* Get all information to connect to the switch send rainbow message or email.
* Extract the switches IP and the ports responsible of the flapping which have been sent in the logs. For that we use the function detect\_port\_flapping(). This function will do the most part of Port Flapping:
* Check if there is more than 30 lines in lastlog\_flapping, if there is the case, we delete the log file. And we return NULL IP Addresses and NULL port numbers
* For each line in the log file we do:
  + We extract the timestamp and the IP address
  + If there is not First\_IP, we put the IP address in the First\_IP, and we put the timestamp in memory for the log of the first ip (last\_time\_first)
  + If there is no Second\_IP and the current IP is not the first IP address, we put the IP address in the Second\_IP and we put the timestamp in memory for the log of the second IP (last\_time\_second)
  + If there is a third IP address, the script clears the log file.
  + To get the port associate to one of the IP Address, the script check if there is LINKSTS in the log, the port number will be the next element in the log.
  + If there is the first port, we copy the port in the variable first\_port, if the first port is already set et the second port is empty we put the port in second\_port.
  + We check if there is DOWN in the log, we don’t take UP log in consideration to don’t make the accounting two times.
  + Now we check if the current ip in the log equals the first or the second IP.
  + Then we calculate the time difference between the current log and the previous log for this IP. If there is less than 10 seconds (for the demo, we can down to 1), the script increments a counter associate to the current IP.
  + When one of the two counters reach at 5, the script returns IP addresses and numbers ports. If there is only one IP address the other IP address return is set to 0 and the port to 1/1/0
  + if counters are less than 5 the script returns NULL IP addresses and NULL port numbers

The main script port\_flapping:

* If both port numbers not equal to 0 , the script continue else the script stops.
* There are now 3 different scenarios: 2 IP addresses, 1st IP address or 2nd IP address, even if the third case couldn’t be reach.
* Send a request by Mail and by Rainbow to fix or not the issue:
  + If the answer is yes, the script sends a ssh packet to disable, and enable the port(s) sent in the logs.
  + If the answer Is yes and save my decision send a ssh packet to disable the port(s) sent in the logs. Then save the answer in the decisions\_save.conf file
  + If the answer is no, the script does nothing, and save the decision in the decisions\_save.conf file.
* Send the log file and a message by the rainbow bot (the bot will be explained in a other part).
* Send the log in attachment and an email to the email address in ALE\_script.conf
* Disable the debugging on the switch(es).

Flowchart:

Diagram

Description automatically generated

## Port Scanning (DDOS) Script

Name: support\_switch\_enable\_qos.py

Rsyslog Trigger: 'ALV4 event: PSCAN'

Last log file : lastlog\_ddos\_ip.json

Log example:

{"@timestamp":"2021-06-04T00:09:18+02:00","type":"syslog\_json","relayip":"10.130.7.251","hostname":"os6860e-core1","message":"<135>Jun 4 00:09:18 OS6860E-Core1 swlogd ipv4 alv4ni DBG1: ALV4 event: PSCAN vrf 0 8.8.4.4","end\_msg":""}

The purpose of this script is to detect a port scanning in a network by analyzed the log file lastlog\_ddos\_ip.json.json, apply a QOS policy to block data access on the IP address responsible of the scan and return back the switch in debug info mode.

Script steps:

* Get all information to connect to the switch send rainbow message or email.
* Extract the switch IP responsible of the scan which have been sent in the logs.
* Extract the attacker IP address with the function extract\_ip\_ddos():
* Check if there is less than 10 seconds between the two last logs
  + If there is less than 10 seconds, the lastlog\_ddos\_ip.json is clearing.
  + If there are less than 2 logs, we do nothing
* If the condition is met, we return the Attacker IP address.
* Send a request by Mail and by Rainbow to fix or not the issue:
  + If the answer is yes, the script sends a ssh packet to disable, and enable the port(s) sent in the logs.
  + If the answer Is yes and save my decision send a ssh packet to disable the port(s) sent in the logs. Then save the answer in the decisions\_save.conf file
  + If the answer is no, the script does nothing, and save the decision in the decisions\_save.conf file.
* Send the log file and a message by the rainbow bot (the bot will be explained in another part).
* Send the log in attachment and an email to the email address in ALE\_script.conf
* Disable the debugging on the switch(es).

Flowchart:

Diagram

Description automatically generated

## Collection Logs (Switch) Script

Name: support\_switch\_get\_log.py

Rsyslog Trigger: Patterns chosen by user during setup.sh

This script collects log by doing a show tech support eng complete on the switch and transfer it by SFTP in the directory /tftpboot/ on the Debian server.

To add commands to get more information specific to the case, there is a section in the python code.

Add l\_switch\_cmd.append("YOUR COMMAND") after the line l\_switch\_cmd = [] in the Get More LOGS section.

## Collection Logs (Stellar AP) Script

Name: support\_AP\_get\_log.py

Rsyslog Trigger: Patterns chosen by user during setup.sh

The script collects log by executing /usr/sbin/take\_snapshot.sh start <Server\_IP> on the AP.

# Add a new script

To add a new script, you have to know which log is involved in the case.

* Add the Rsyslog part in the Setup.sh file here an example for the loop:
  + Add :

Template (name=\"devicelogloop\" type=\"string\"

string=\"/var/log/devices/lastlog\_loop.json\")

in the Template definition section

* + Change devicelogloop and var/log/devices/lastlog\_loop.json by what you want
  + Add :

if \$msg contains 'slnHwlrnCbkHandler' and \$msg contains 'port' and \$msg contains 'bcmd' then { \$RepeatedMsgReduction on

action(type=\"omfile\" DynaFile=\"deviceloghistory\" template=\"json\_syslog\" DirCreateMode=\"0755\" FileCreateMode=\"0755\")

action(type=\"omfile\" DynaFile=\"devicelogloop\" template=\"json\_syslog\" DirCreateMode=\"0755\" FileCreateMode=\"0755\")

action(type=\"omprog\" binary=\"$dir/support\_switch\_port\_disable.py\" queue.type=\"LinkedList\" queue.size=\"1\" queue.workerThreads=\"1\")

#queue.type=\"LinkedList\" queue.size=\"1\" queue.discardmark=\"1\" queue.TimeoutActionCompletion=\"2000\") stop}

* + Change slnHwlrnCbkHandler /port/bcmd by what specific keyword in the log.
  + Change devicelogloop by what you put in the first step
  + Change the name of the python script
* Put you script in the /opt/ALE\_Script folder
* Do chmod +x “script name”

End of the document