

Technical Support DATA (EMEA)

**System Functional and Design Specification**

**Preventive Maintenance**

Author(s): Raphael Voyer, Jonathan Trebaol

Revision: 0.1

Date: 10/06/2021

**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 6](#_Toc76555045)

[1.1 Purpose 6](#_Toc76555046)

[1.2 Scope 6](#_Toc76555047)

[1.3 Intended Audience 6](#_Toc76555048)

[2 FUNCTIONAL DESCRIPTION 7](#_Toc76555049)

[2.1 Basic Overview 7](#_Toc76555050)

[2.2 Platform Supported 7](#_Toc76555051)

[2.3 Design Constraints. 8](#_Toc76555052)

[2.3.1 Software Limitations 8](#_Toc76555053)

[2.3.2 Hardware Limitations 8](#_Toc76555054)

[2.4 Assumptions and Dependencies 8](#_Toc76555059)

[3 SYSTEM REQUIREMENT SPECIFICATIONS 9](#_Toc76555061)

[3.1 Configuration Requirements 9](#_Toc76555062)

[3.1.1 Python Requirements 9](#_Toc76555063)

[3.1.2 TFTP Requirements 9](#_Toc76555071)

[4 DEBIAN SERVER INITIALIZATION 9](#_Toc76555072)

[4.1 Introduction 9](#_Toc76555073)

[4.2 Flowchart 9](#_Toc76555074)

[4.3 Services 9](#_Toc76555075)

[4.3.1 TFTP 9](#_Toc76555076)

[4.3.2 Rsyslog 9](#_Toc76555077)

[4.3.3 Logrotate 10](#_Toc76555078)

[4.3.4 Iptables 10](#_Toc76555079)

[5 UTILS PYTHON SCRIPTS 11](#_Toc76555080)

[5.1 Introduction 11](#_Toc76555081)

[5.2 Support Tools Script 11](#_Toc76555082)

[5.3 Active Output Socket 11](#_Toc76555083)

[5.4 Send Notification 11](#_Toc76555084)

[5.5 Requests Handler 11](#_Toc76555085)

[5.6 Web Receiver 12](#_Toc76555086)

[6 USE CASE PYTHON SCRIPTS 12](#_Toc76555087)

[6.1 L2 Loop Script 12](#_Toc76555088)

[6.2 Port Flapping Script 12](#_Toc76555089)

[6.3 Port Scanning (DDOS) Script 12](#_Toc76555090)

[6.4 Collection Logs (Switch) Script 12](#_Toc76555091)

[6.5 Collection Logs (Stallar AP) Script 12](#_Toc76555092)

**REVISION HISTORY**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Revision | Date | Agile Revision | Update By | Revision Description | Status |
| 0.1 | 16/06/2021 |  | Raphael Voyer | Initial document, | Draft |

# INTRODUCTION

## Purpose

Purpose The document describes the Software Requirements Specification and Functional Specification of Preventive Maintenance for AOS 8.x

## Scope

The scope of this document is to present the complete set of requirements and functional specifications for Preventive Maintenance.

## Intended Audience

This document is intended for the following:

* Support organization providing end-user support.







# FUNCTIONAL DESCRIPTION

## Basic Overview

Preventive maintenance allows the automation of troubleshooting on OmniSwitch AOS 8.x, Stellar APs equipment, and a notification function either by email or by Rainbow can be configured. It consists of a Debian server that will receive logs of the various devices on the network. All logs will be filtered by Rsyslog. Rsyslog will both detect one or more patterns within the logs, and then store them and run a script based on the received log. All scripts for resolutions are developed in Python

Here an example of loop detection and resolution:

1. Server receives of a syslog message containing the pattern "Buffer list is empty"

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:  
• **[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)**[:](#_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 etc...

• **[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 conducts the sending and receiving requests by notifications (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

For the first release of Preventive Maintenance, in order to fully configure the Debian server, bash script under the name of Setup.sh has been developed. This allows you to configure the following:

• Configuration of notifications (Rainbow/Email)

• Pattern for collecting switchs logs

• Credentials of Switchs

• Credentials of Stellar access points

• Pattern for collecting AP logs

• Subnets authorized to send logs

• Installation of Python3 and outbuildings

• Rsyslog configuration

• Logrotate configuration

• Configuration of iptables (not yet implemented)

• TFTP Server Configuration

• Active socket output on switches

• Create /opt/ALE\_Script directory

To save all theses information in ALE\_script.conf .

Example:

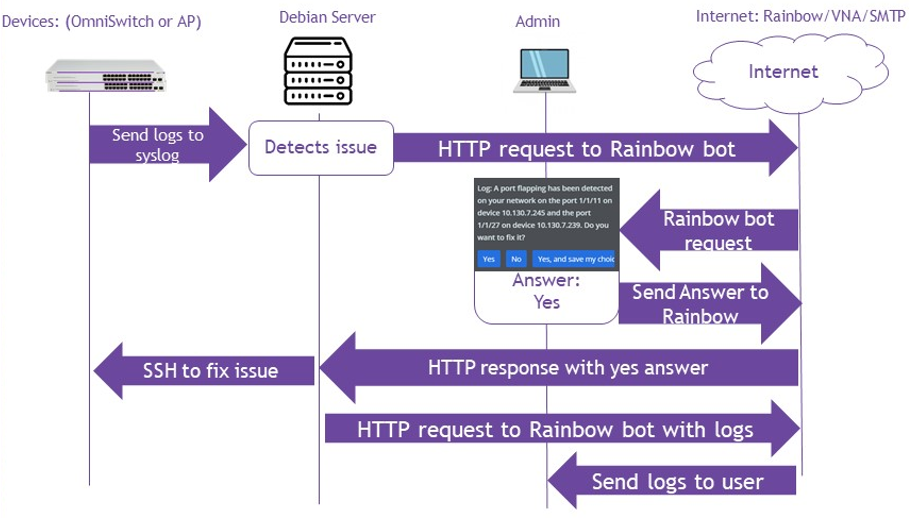
admin,switch,email1@al-enterprise.com;email2@al-enterprise.com,59fe823c15xxxxxxxxxxxx4feef59a@openrainbow.com,sender@gmail.com,pass\_mail\_sender,10.130.7.14,support,Letacla01\*,Letacla01\*,6573953192,,,

This structure file is defined by boxes, each box is separate by ‘,’.

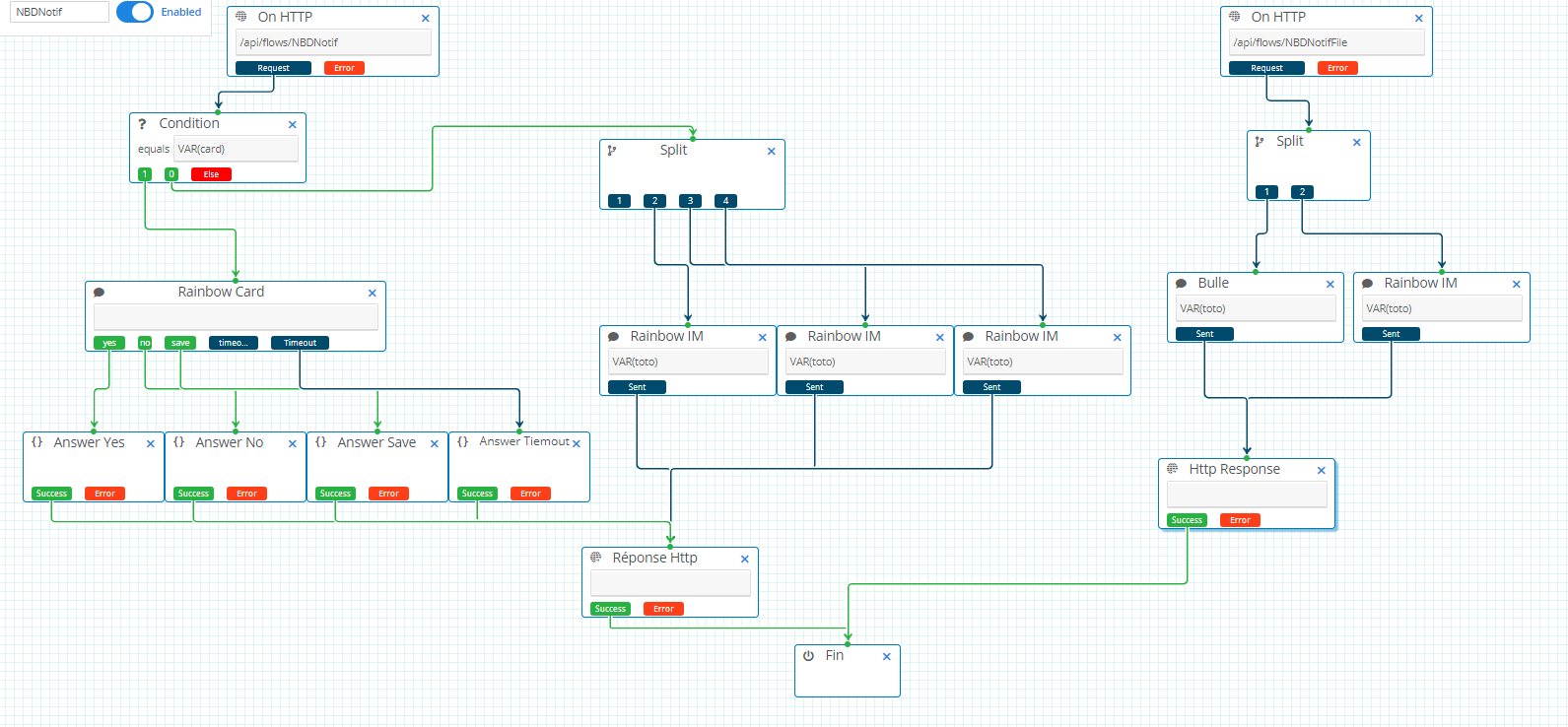
Here the compositions of each box:

* Switches login
* Switches password
* Admin emails( if more than 1 email separate each by ‘;’)
* Rainbow JID
* Email use to send requests ,logs, notifications
* Ip address of the Debian Server.
* If AP are use , AP login
* If AP are use, AP password
* If AP are use, AP technical support code
* Client ID, which is a random number between 0000000000 and 9999999999 create at the execution of Setup.sh. The number function is to be sure the answer from a request is the good one. (this number is deeply explain in the part 5.6)

Rainbow request:



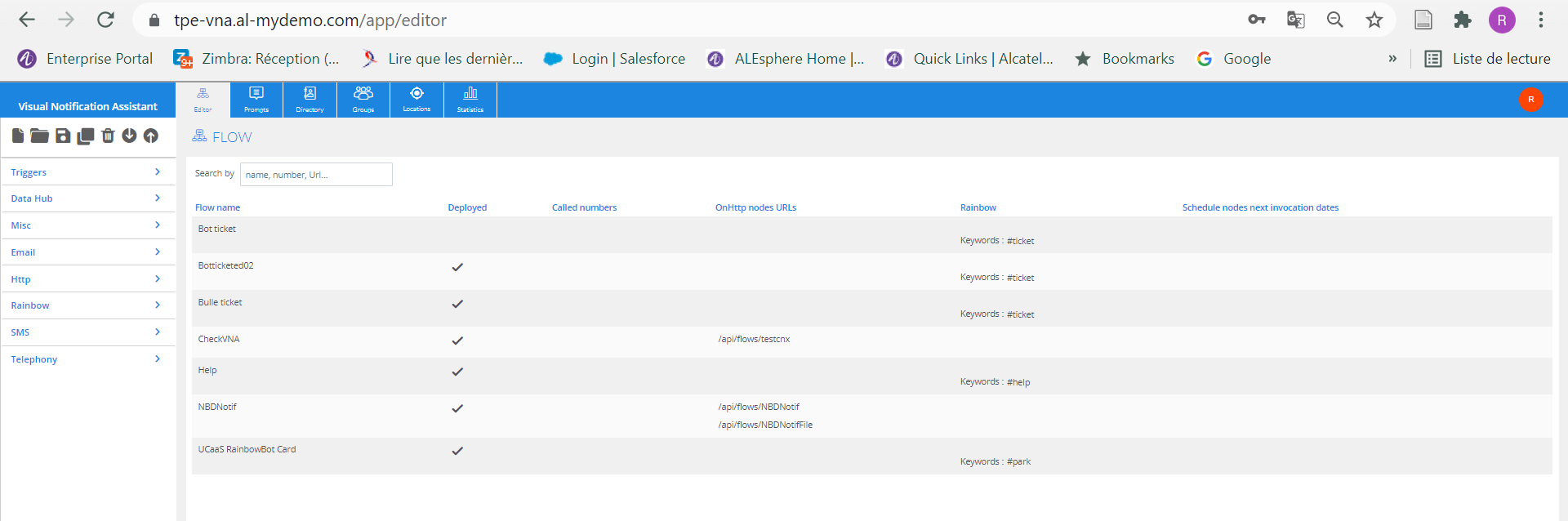
VNA permit the creation, and the configuration of the Rainbow bot. The purpose is simple, everything is configurable in the form of a block that takes each one a function, it is then enough to put it in the following or a parallel according to what we want to do with the bot , here is a flow created on VNA for the Preventive Maintenance bot:



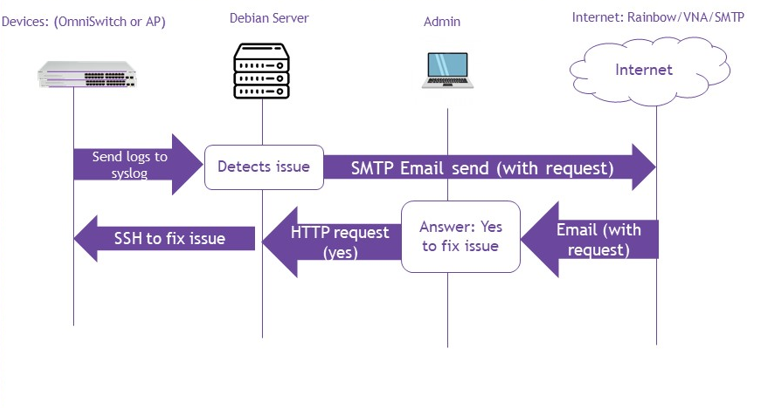
To access and configure the VNA workflow, go to this url : <https://tpe-vna.al-mydemo.com/app/login>

If you don’t have account, ask to PEYREBESSE Thierry.

Then, click on editor and NBDNotif:



Email request:



## Platform Supported

The proposed Preventive Maintenance functionality shall be supported on Omniswitch AOS 8.X and AP Stellar (all models at the date 28/07/2021)

## Prerequisites

This chapter captures the requirements for Preventive Maintenance.

Supported OS: Debian, Raspbian.

Debian Server: 9.13

SSH: OpenSSH\_7.4p1

Python Version: Python 3.5.3 and above

## Design Constraints.

### Software Limitations

* Unknown

### Hardware Limitations

* Unknown

# USER GUIDE

Creation of Preventive Maintenance Environment:

1. Download the Git folder Scripts\_ALE
2. Cd /Script\_ALE
3. Sudo ./Setup.sh ( follow configuration steps)

Verification of the Preventive Maintenance Environment:

1. Ls /opt/ 🡺 folder ALE\_Script created
2. Cd /opt/ALE\_Script
3. Ls 🡺

Before use Preventive Maintenance scripts:

* + - 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

After use scripts:

* device\_catalog.conf (IP addresses extract from Device.csv)
* decisions\_save.conf (permit the record of decision for a particular case, example: loop with port 1/1/4 on devices 10 192.168.80.27 always fix the issue : *192.168.80.27,1/1/4,loop,always)*
* configqos (the qos configuration to block internet from an attacker, works with the support\_switch\_enable\_qos.py script.

# 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). The purpose of the Setup.sh script is to initialize all the services that will be useful for the proper operation of the Preventive Maintenance functionalities. To run Preventive Maintenance requires service that are not present on a Debian server at installation. It is also necessary to be able to configure these different services. A security is also added to prevent bad entries, checking the form of the mail or an IP address for example.

## Services

All services described below are fully installed and configured by the Setup.sh script. It is not necessary to modify the configuration files, in order to update the Rsyslog rules based on pattern, please re-execute the Setup.sh script

### TFTP

A TFTP Server is installed on the server, the default director is set to /tftpboot/. This service is required for uploading logs from Stellar APs. TFTP Server is used by following functions:

* Switchs (show tech-support eng complete)
* AP Stellar (take\_snapshot.sh start).

Commands to verify the configuration:

* systemctl status tftpd-hpa (the services must be in running mode)
* cat /etc/default/tftpd-hpa , the file configuration must be:

# /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 and based on patterns execute the following tasks:

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

- forward the logs to specific files for processing /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 services must be in running mode)

It will also create another temporary file depending on the case, in order to be able to handle more easily the latest logs received:

• lastlog.json

• lastlog\_ddos.json

• lastlog\_ddos\_ip.json

• lastlog\_flapping.json

• lastlog\_loop.json

These various files will be described in the relevant parts of the report.

Rsyslog will also allow depending on the patterns detected in the logs to run the corresponding python scripts.

##### Flowchart

Here the definition of Rsyslog rules and actions executed:

Diagram

Description automatically generated

### Logrotate

Logrotate limits the size of log files 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

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

Logrotate is configure to work on /var/log/devices/ files with the extension .json or .log, and compress them daily into .gz .

Find the config file at the path /etc/logrotate.d/Rsyslog

To execute the rotation: logrotate -d /etc/logrotate.d/rsyslog

### Iptables

Not yet implemented

### Python Utilities

Preventive Maintenance feature needs some packages to be install on the Debian server, which are install during the 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.

Pystfp: Use for the stfp file transfer with python code

Flask: Use for the creation and configuration of a webserver with python code.

Tftpd-hpa: Use for the creation and configuration of a tftp server on the Debian server.

# UTILS PYTHON SCRIPTS

## Introduction

The use of ‘utils’ scripts allows multiple scripts using the same functions. For example, all scripts use the utils scripts send\_notification and support\_tools. It also makes it easier to find the code by splitting it and classifying it into several files. (requests handler, support tools)

## Support Tools Script

Name: support\_tools.py

Rather all the tools allowing the automation of network support tasks.

There are the functions that allow you to perform actions on the switches.

There are also all the functions to detect incidents.

This script allows the factorization of the code in order to reuse the s functions that can be identical on the different scripts.

Functions:

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

send the command “*swlog appid bcmd subapp 3 level debug2*” to the switch in parameters

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

send the command “*swlog appid bcmd subapp all level info*” to the switch in parameters

**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.

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

This function disables the debugging level 3 on the switch put in arguments,to get more details in log.

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

send the command “*interfaces port admin-state disable*" to the switch and port in parameters.

**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 parameters.

**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 parameters.

**file\_setup\_qos**(ipadd):

Create qos configuration to block the attacker data access.

**file\_unset\_qos**(ipadd):

Create the qos configuration to deblock the attacker data access.

**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 (more than 10 log in 2 seconds)

**detect\_port\_flapping**():

This function detects if there is flapping in the log.

If there is more than 5 logs with 10 seconds apart between each, there is flapping.(10 seconds is for the demo, we can down to 1)

**save\_attachment**(ipadd):

copy the 30th last logs of the equipment in parameter in the /var/log/devices/attachment.log file.

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

**extract\_ip\_port**():

This function collects the IP address of the switch, in the log.

if, this is a loop log, 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 check if there is a real ddos or not.

**check\_timestamp**():

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

**get\_credentials**():

This founction allows to collect all the credentials of the switchs,

but also, credentials of the mail sender, and information useful to send notification.

**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 all the information about 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 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 new instruction to be recorded given by the user on Rainbow.

## Active Output Socket

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

Rather all functions to send email or rainbow message.

Email functions can be a request to act or not on the issue. Feature handle attachments files.

Rainbow function Rainbow functions can be a request to act or not on the issue. Feature handle text attachments files.

## Web Receiver

Name: support\_web\_receiver\_class.py

Permit the setup of the web server, with goods keys and values in url parameters.

For example :

when the web server is started, there is only 3 pages 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>

To ensure security, the other parameters to access at the good page is the client id and case id.

When the server receives an answer by access on the url, the webserver stops.

If there is no access on the web server after 60 seconds the answer will be automatically yes.

This script is only call by the request handler, which permit to manage the whole part of notification requests.

## Requests Handler

Name: support\_response\_handler.py

Conductor of the request feature.

The purpose of this script, when an issue occurred, we have to send a request by mail and/or by rainbow to the Admin. For that we must be sure the answer give by the Admin is not for another issue occurred at the same time. For that we need to have the client ID create during the Setup.sh execution and a Case ID create at the execution of support\_response\_handler.py.

Then we have to execute 2 services at the same time for the same case, send request email and send rainbow request. For that, we use thread (which permit to execute multiple tasks at the same time)

The Script starts the web service, with the good parameters for the case (with ID numbers)

The script sends the email with the good ID numbers

The script then 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

To work the script needs the save\_decision.conf file, which will register the cases, that the administrator already fix and wanted to save or the case that the administrator don’t want to fix.

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 recorded 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 scripts get the id of the client, and create an id for the case, which is a random number between 0000000000 and 9999999999.
* Creation of the web server (Web Receiver)
* Send request by email
* Send request by Rainbow
* Wait for the turn 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 must send request to Admin.

# USE CASE PYTHON SCRIPTS

The pattern in the logs received from Alcatel equipment alone are not enough to detect whether this is a real network issue.

For this purpose, functions allowing the analysis of the latest logs with the same received pattern is done (on the frequency of received messages, the quantity of received messages and thanks to Timestamps).

The scripts also allow the sending of commands to the equipment, which will allow the resolution of incidents.

## 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. Indeed, if the debug mode is set on ‘info’. All log received on the server log will be Buffer is not empty. To get the log with more detail on our server log we need to enter the switch in debug2 mode.

To do that we use the enable\_debugging function which uses sshpass to enter the command on the switch.

When the script is executed, a ssh command “*swlog appid bcmd subapp 3 level debug2*” is send to the device which sent the log.

This commends permit the reception of more detail logs, then the rsyslog can detect which case are responsible of these logs.

After the switch is 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

Log example:

When the script is executed, a ssh command “*swlog appid ipv4 subapp all level debug3*” is send to the device which sent the log.

This commends permit the reception of more detail logs, then the rsyslog can detect which case are responsible of these logs.

## 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":""}*

The purpose of this script is to detect a L2 loop in a network by analyzed the log file lastlog\_loop.json, shut down the port which responsible of the loop and return back the switch in debug mode info.

First step, the script gets all information to connect to the switch send rainbow message or email.

Next, the script extracts the switch IP and the port responsible of the loop which have been sent in the logs.

Script Steps:

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

To do that, 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 in it.
* 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 (else, there is an error due to second changes 60 to 0)
* 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 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 sends 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 a email to the email address 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.

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":""}*

The purpose of 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 a email to the email address in ALE\_script.conf
* Disable the debugging on the switch(es).

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).

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 collect log by doing a “show tech support complete eng” on the switch ans trasfert it by SFTP in the directory /tftpboot on the Debian server.

## Collection Logs (Stellar AP) Script

Name: support\_AP\_get\_log.py

Rsyslog Trigger: Patterns chosen by user during setup.sh

The script collect log by doing “/usr/sbin/take\_snapshot.sh start <Debian IP>” on the AP .