Automate Catalyst 9000 Switches Using Python Scripts

Contents

Introduction

This document describes how to extend EEM with Python scripts for automating configuration and data collection on Catalyst 9000 switches.

Prerequisites

Requirements

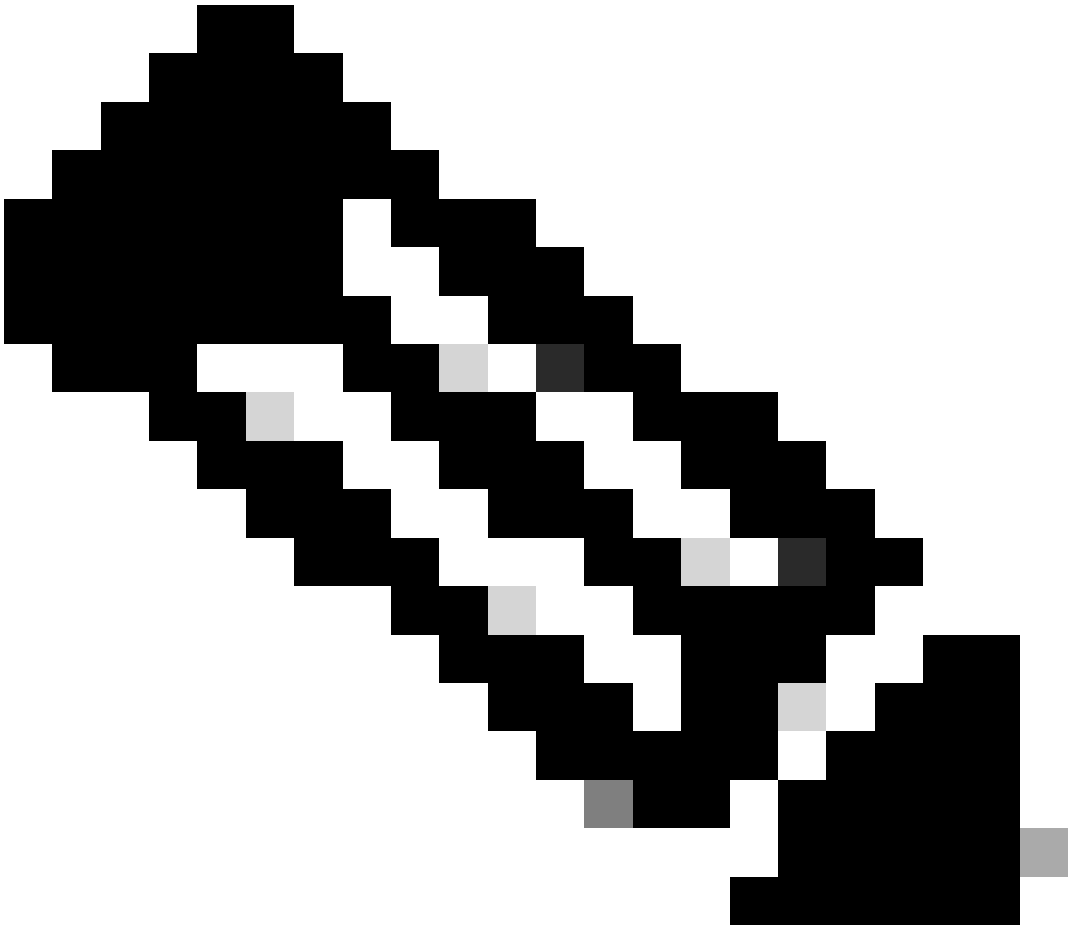
Cisco recommends that you have knowledge of and familiarity with these topics:

- Cisco IOS® and Cisco IOS® XE EEM
- Application Hosting and Guest Shell
- Python scripting
- Linux commands

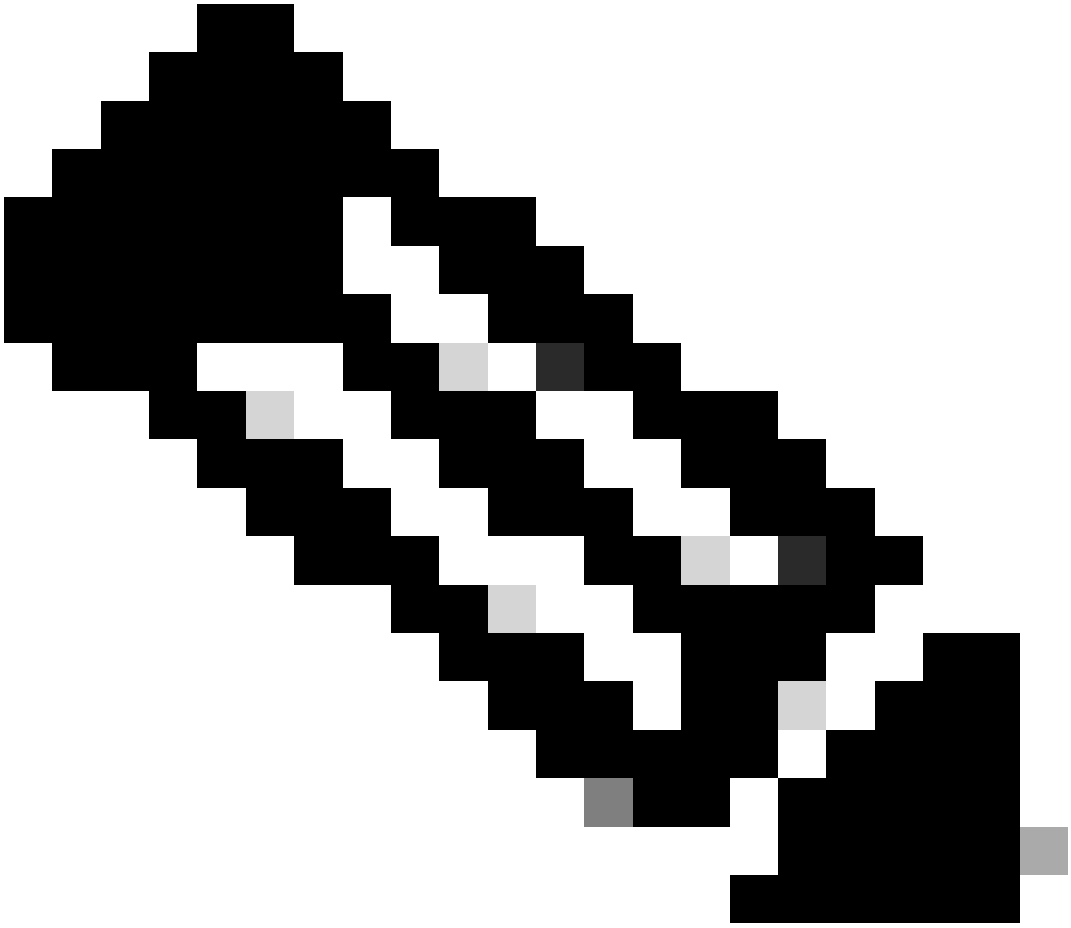
Components Used

The information in this document is based on these software and hardware versions:

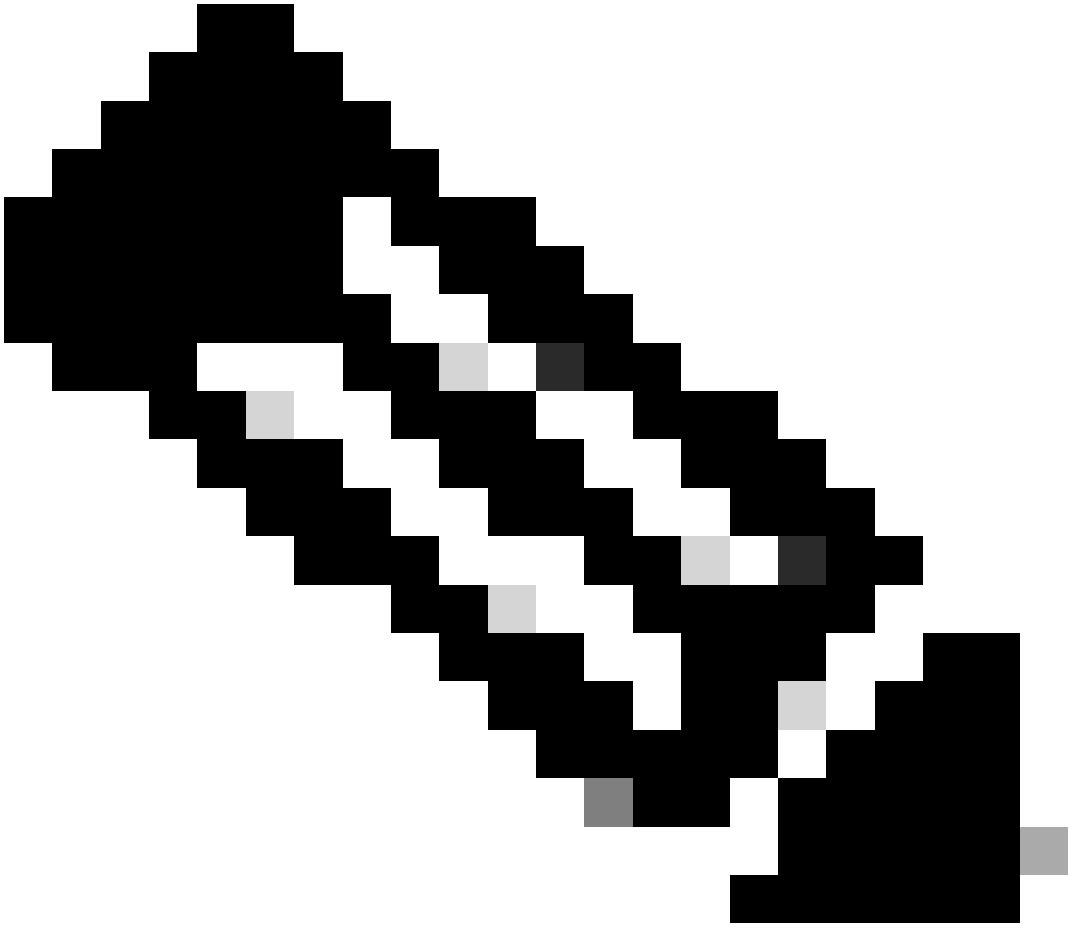
- Catalyst 9200
- Catalyst 9300
- Catalyst 9400
- Catalyst 9500
- Catalyst 9600
- Cisco IOS XE 17.9.1 and later versions



Note: Consult the appropriate configuration guide for the commands that are used in order to enable these features on other Cisco platforms.



Note: Catalyst 9200L switches do not support Guest Shell.



Note: These scripts are not supported by Cisco TAC and are provided on an as-is basis for educational purposes.

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. If your network is live, ensure that you understand the potential impact of any command.

Conventions

Refer to Cisco Technical Tips Conventions for information on document conventions.

Background Information

Guest Shell and Python Scripts

Application hosting on the Cisco Catalyst 9000 family of switches introduces innovation opportunities for partners and developers, as network devices can be merged with an application runtime environment.

It supports containerized applications, providing complete isolation from the main operating system and the Cisco IOS XE Kernel. This separation ensures that resource allocations for hosted applications are distinct from the core routing and switching functions.

The Application hosting infrastructure for Cisco IOS XE devices is known as IOx (Cisco IOS + Linux), which facilitates the hosting of applications and services developed by Cisco, partners, and third-party developers on network devices, ensuring seamless integration across diverse hardware platforms.

The Guest Shell, a specialized container deployment, exemplifies an application beneficial for system deployment.

The Guest Shell offers a virtualized Linux-based environment designed to run custom Linux applications, including Python, to enable automated control and management of Cisco devices. The Guest Shell container allows users to run scripts and apps within the system. Specifically, on Intel x86 platforms, the Guest Shell container is a Linux Container (LXC) with a CentOS 8.0 minimal root file system. In Cisco IOS XE Amsterdam 17.3.1 and later releases, only Python V3.6 is supported. Additional Python libraries can be installed during runtime using the Yum utility in CentOS 8.0. Python packages can also be installed or updated using Pip Install Packages (PIP).

Guest Shell includes a Python Application Programming Interface (API), which allows to run Cisco IOS XE commands using the Python CLI module. This way, Python scripts enhance automation capabilities, providing network engineers with a versatile tool to develop scripts for automating configuration and data collection tasks. While these scripts can be run manually through the CLI, they can also be employed alongside EEM scripts to respond to specific events, such as syslog messages, interface events, or command runs. Practically, any event that can trigger an EEM script can also be used to trigger a Python script, expanding the automation potential within Cisco Catalyst 9000 switches.

When Guest Shell is installed, a guest-share directory is automatically created in the flash file system. This is the file system that can be accessed from the Python scripts and Guest Shell. To ensure proper synchronization when using stacking, keep this folder under 50 MB.

Benefits of Using Python with EEM Scripts

- Python extends the automation capabilities of EEM scripts by allowing complex logic (such as regular expressions, loops, and matches) to be handled within the Python script. This capability provides an opportunity to create more powerful EEM scripts.
- As a well-known programming language, Python lowers the barrier to entry for network engineers who wish to automate Cisco IOS XE devices. Additionally, it offers maintainability and readability.
- Python also provides error-handling capabilities as well as a powerful standard library.

Considerations of Using Python with EEM Scripts

- Guest Shell is not enabled by default, so it needs to be enabled before Python scripts can be run.
- Python scripts cannot be created directly in the CLI; they need to be developed first in a development environment and then copied to the flash memory of the switch.

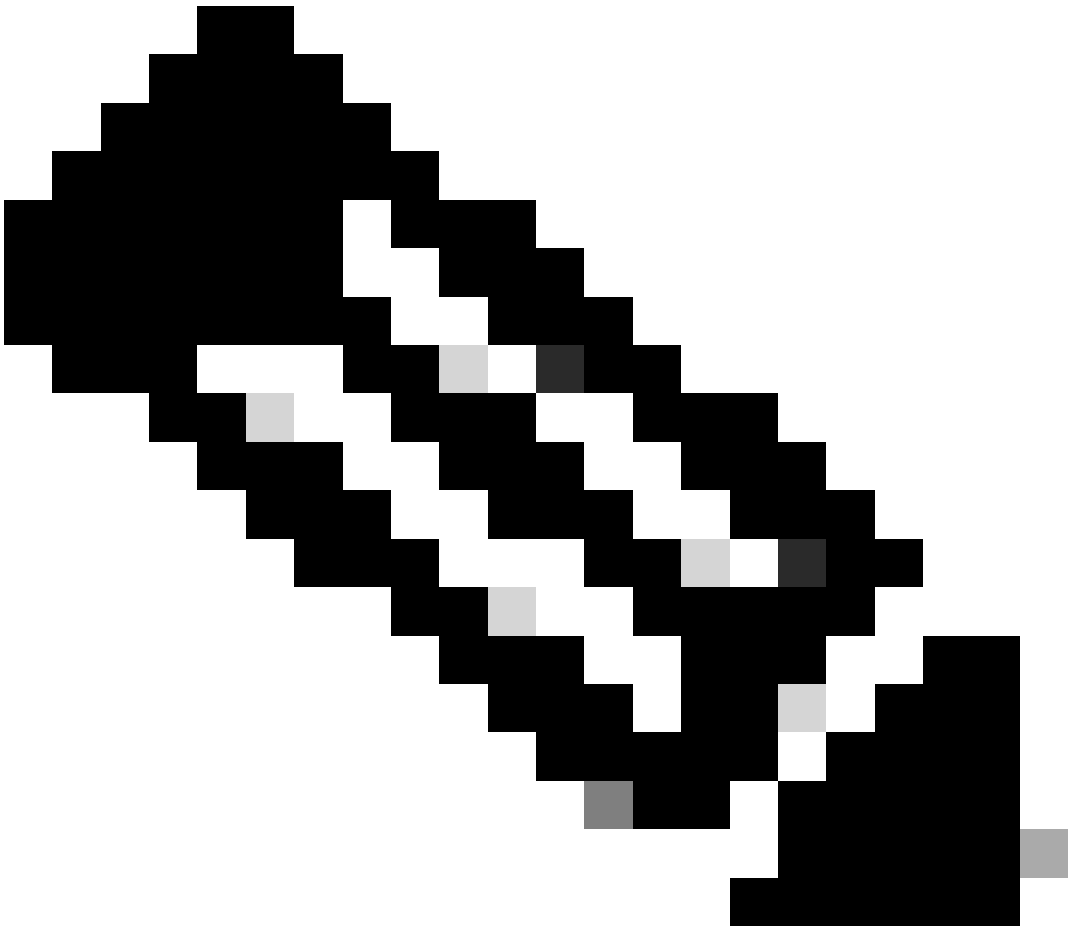
SELinux in Cisco IOS XE

Starting with Cisco IOS XE 17.8.1, support for Security-Enhanced Linux (SELinux) was introduced to enforce security with policies that govern how processes, users, and files interact with each other. The SELinux policy defines which actions and resources are allowed to be accessed by a process or user. A

violation can occur when a user or process tries to perform an action not permitted by the policy, for example, accessing a resource or running a command. SELinux can operate in 2 different modes:

1. Permissive mode: SELinux does not enforce any policies. However, it still logs any violations as if they were being denied.

2. Enforcing: SELinux actively enforces the security policies on the system. If an action violates the SELinux policy, the action is denied and logged.



> **Note**: The default mode was set to permissive when it was introduced in Cisco IOS XE 17.8.1. However, starting with version 17.14.1, SELinux is enabled in enforcement mode.

When using Guest Shell, access to some resources can be denied when using enforcing mode. If when trying to perform an action using Guest Shell or a Python script results in a permission denied error, similar to this log:

```
*Jan 21 13:22:01: %SELINUX-1-VIOLATION: Chassis 1 R0/0: audispd: type=AVC msg=audit(1738074795.448:198)
```

To verify if a script is being denied by SELinux, use the command show platform software audit summary to check if the Denial Counts are increasing. Additionally, show platform software audit all displays a log of actions blocked by SELinux. The Access Vector Cache (AVC) is the mechanism used to record access control decisions in SELinux, so when using this command, look for logs that begin with type=AVC.

If a script is being denied and blocked, SELinux can be set to permissive mode using the command set platform software selinux permissive. This change is not stored in the running or startup configuration, so after a reload, the mode reverts to enforcing. Therefore, each time the switch reloads, this change needs to be reapplied. The change can be verified using show platform software selinux.

Configure

To automate tasks on the switch using EEM and Python scripts, follow these steps:

1. Enable the **Guest Shell**.
2. Copy the **Python script** to /flash/guest-share/ directory. You can use any copy mechanism available in Cisco IOS XE, such as SCP, FTP, or the FIle Manager in the WebUI. Once the Python script is in the flash memory, you can run it using the command guestshell run python3 /flash/guest-share/cat9k_script.py.
3. Configure an **EEM script** that runs the Python script. This setup allows the Python script to be triggered using any of the multiple event detectors provided by EEM scripts, such as a syslog message, a CLI pattern, and a Cron scheduler.

In this section, Step 1 is explained. The next section provides examples demonstrating how to implement Steps 2 and 3.

Enable the Guest Shell with a Static IP Address

Follow this process to enable the Guest Shell:

1. Enable **IOx**.

<#root>

```
Switch#conf t
Switch(config)#iox
Switch(config)#
*Feb 17 18:13:24.440: %UICFGEXP-6-SERVER_NOTIFIED_START: Switch 1 R0/0: psd: Server iox has been r
*Feb 17 18:13:28.797: %IOX-3-IOX_RESTARTABITLITY: Switch 1 R0/0: run_ioxn_caf: Stack is in N+1 mod

*Feb 17 18:13:36.069: %IM-6-IOX_ENABLEMENT: Switch 1 R0/0: ioxman: IOX is ready.
```

2. Configure the **Application Hosting Network** for the Guest Shell. This example uses the AppGigabitEthernet interface to provide network access; however, the Management interface (Gi0/0) can also be used.

```
Switch(config)#int appgig1/0/1
Switch(config-if)#switchport mode trunk
Switch(config-if)#switchport trunk allowed vlan 20

Switch(config)#app-hosting appid guestshell
```

```
Switch(config-app-hosting)#app-vnic appGigabitEthernet trunk
Switch(config-config-app-hosting-trunk)#vlan 20 guest-interface 0
Switch(config-config-app-hosting-vlan-access-ip)#guest-ipaddress 10.20.1.2 netmask 255.255.255.0
Switch(config-config-app-hosting-vlan-access-ip)#exit
Switch(config-config-app-hosting-trunk)#exit
Switch(config-app-hosting)#app-default-gateway 10.20.1.1 guest-interface 0
Switch(config-app-hosting)#name-server0 10.31.104.74
Switch(config-app-hosting)#end
```

3. Enable the **Guest Shell**.

```
<#root>

Switch#guestshell enable
Interface will be selected if configured in app-hosting
Please wait for completion
guestshell installed successfully
Current state is: DEPLOYED
guestshell activated successfully
Current state is: ACTIVATED
guestshell started successfully
Current state is: RUNNING

Guestshell enabled successfully
```

4. Validate the **Guest Shell**. This example validates that there is reachability with the default gateway as well as with cisco.com. Also, validate that **Python 3** can be run from the Guest Shell.

```
<#root>

! Validate that the Guest Shell is running.
Switch#

show app-hosting list


App id                                   State
------------------------------------------------------------

guestshell


RUNNING


Switch#

guestshell run bash

[guestshell@guestshell ~]$

! Validate that the IP address of the Guest Shell is configured correctly.
[guestshell@guestshell ~]$

sudo ifconfig

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
```

```
inet 10.20.1.2  netmask 255.255.255.0

  broadcast 10.20.1.255
        inet6 fe80::5054:ddff:fe61:24c7  prefixlen 64  scopeid 0x20
        ether 52:54:dd:61:24:c7  txqueuelen 1000  (Ethernet)
        RX packets 23  bytes 1524 (1.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9  bytes 726 (726.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 177  bytes 34754 (33.9 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 177  bytes 34754 (33.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

! Validate reachability to the default gateway and ensure that DNS is resolving correctly.
[guestshell@guestshell ~]$

ping 10.20.1.1

PING 10.20.1.1 (10.20.1.1) 56(84) bytes of data.
64 bytes from 10.20.1.1: icmp_seq=2 ttl=254 time=0.537 ms
64 bytes from 10.20.1.1: icmp_seq=3 ttl=254 time=0.537 ms
64 bytes from 10.20.1.1: icmp_seq=4 ttl=254 time=0.532 ms
64 bytes from 10.20.1.1: icmp_seq=5 ttl=254 time=0.574 ms
64 bytes from 10.20.1.1: icmp_seq=6 ttl=254 time=0.590 ms
^C
--- 10.20.1.1 ping statistics ---
6 packets transmitted, 5 received, 16.6667% packet loss, time 5129ms
rtt min/avg/max/mdev = 0.532/0.554/0.590/0.023 ms

[guestshell@guestshell ~]$

ping cisco.com

PING cisco.com (X.X.X.X) 56(84) bytes of data.
64 bytes from www1.cisco.com (X.X.X.X): icmp_seq=1 ttl=237 time=125 ms
64 bytes from www1.cisco.com (X.X.X.X): icmp_seq=2 ttl=237 time=125 ms
^C
--- cisco.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 124.937/125.141/125.345/0.204 ms

! Validate the Python version.
[guestshell@guestshell ~]$

python3 --version

Python 3.6.8


! Run Python commands within the Guest Shell.
[guestshell@guestshell ~]$

python3

Python 3.6.8 (default, Dec 22 2020, 19:04:08)
[GCC 8.4.1 20200928 (Red Hat 8.4.1-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>>

print("Cisco")
Cisco

>>> exit()
[guestshell@guestshell ~]$

[guestshell@guestshell ~]$ exit
exit

Switch#
```

Enable the Guest Shell with a DHCP IP Address

Usually, the Guest Shell is configured with a static IP address because the Guest Shell container does not have the DHCP client service by default. If it is necessary for the Guest Shell to request an IP address dynamically, the DHCP client service needs to be installed. Follow this process:

1. Follow the steps to enable the Guest Shell with a static IP address. However, this time, do not assign the IP address in the app-hosting configuration during step 2. Instead, use this configuration:

```
Switch(config)#int appgig1/0/1
Switch(config-if)#switchport mode trunk
Switch(config-if)#switchport trunk allowed vlan 20

Switch(config)#app-hosting appid guestshell
Switch(config-app-hosting)#app-vnic appGigabitEthernet trunk
Switch(config-config-app-hosting-trunk)#vlan 20 guest-interface 0
Switch(config-app-hosting)#end
```

2. The DHCP client can be installed using the Yum utility with the command sudo yum install dhcp-client. However, repositories for CentOS Stream 8 have been decommissioned. To address this, the DHCP client packages can be downloaded and installed manually. On a PC, download these packages from the CentOS Stream 8 vault and package them into a tar file.

    - bind-export-libs-9.11.36-13.el8.x86_64.rpm
    - dhcp-client-4.3.6-50.el8.x86_64.rpm
    - dhcp-common-4.3.6-50.el8.noarch.rpm
    - dhcp-libs-4.3.6-50.el8.x86_64.rpm

```
[cisco@CISCO-PC guestshell-packages] % tar -cf dhcp-client.tar bind-export-libs-9.11.36-13.el8.x86
```

3. Copy the dhcp-client.tar file to /flash/guest-share/ directory in the switch.

4. Enter a **Guest Shell bash session** and run the **Linux** commands to install the DHCP client and request an IP address.

<#root>

```
513E.D.02-C9300X-12Y-A-17#

guestshell run bash


[guestshell@guestshell ~]$

sudo ifconfig

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536

<--- no eth0 interface

        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 149  bytes 32462 (31.7 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 149  bytes 32462 (31.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

! Unpack the packages needed for the DHCP client service.
[guestshell@guestshell ~]$

tar -xf /flash/guest-share/dhcp-client.tar

tar: Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.quarantine'
tar: Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.metadata:kMDItemWhereFro
tar: Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.macl'
tar: Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.quarantine'
tar: Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.metadata:kMDItemWhereFro
tar: Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.macl'
tar: Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.quarantine'
tar: Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.metadata:kMDItemWhereFro
tar: Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.macl'
tar: Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.quarantine'
tar: Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.metadata:kMDItemWhereFro
tar: Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.macl'

[guestshell@guestshell ~]$

ls

bind-export-libs-9.11.36-13.el8.x86_64.rpm  dhcp-common-4.3.6-50.el8.noarch.rpm
dhcp-client-4.3.6-50.el8.x86_64.rpm         dhcp-libs-4.3.6-50.el8.x86_64.rpm

! Install the packages using DNF.
[guestshell@guestshell ~]$

sudo dnf -y --disablerepo=* localinstall *.rpm

Warning: failed loading '/etc/yum.repos.d/CentOS-Base.repo', skipping.
Dependencies resolved.
==========================================================================================================
 Package                   Architecture    Version                      Repository            Siz
==========================================================================================================
Installing:
 bind-export-libs          x86_64          32:9.11.36-13.el8            @commandline          1.1
 dhcp-client               x86_64          12:4.3.6-50.el8              @commandline          319
 dhcp-common               noarch          12:4.3.6-50.el8              @commandline          208
 dhcp-libs                 x86_64          12:4.3.6-50.el8              @commandline          148

Transaction Summary
==========================================================================================================
Install  4 Packages
```

```
Total size: 1.8 M
Installed size: 3.9 M
Downloading Packages:
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                                                                1/
  Installing       : dhcp-libs-12:4.3.6-50.el8.x86_64                                                               1/
  Installing       : dhcp-common-12:4.3.6-50.el8.noarch                                                             2/
  Installing       : bind-export-libs-32:9.11.36-13.el8.x86_64                                                      3/
  Running scriptlet: bind-export-libs-32:9.11.36-13.el8.x86_64                                                      3/
  Installing       : dhcp-client-12:4.3.6-50.el8.x86_64                                                             4/
  Running scriptlet: dhcp-client-12:4.3.6-50.el8.x86_64                                                             4/
  Verifying        : bind-export-libs-32:9.11.36-13.el8.x86_64                                                      1/
  Verifying        : dhcp-client-12:4.3.6-50.el8.x86_64                                                             2/
  Verifying        : dhcp-common-12:4.3.6-50.el8.noarch                                                             3/
  Verifying        : dhcp-libs-12:4.3.6-50.el8.x86_64                                                               4/

Installed:
  bind-export-libs-32:9.11.36-13.el8.x86_64           dhcp-client-12:4.3.6-50.el8.x86_64
  dhcp-common-12:4.3.6-50.el8.noarch                  dhcp-libs-12:4.3.6-50.el8.x86_64

Complete!

! Request a DHCP IP address for eth0.
[guestshell@guestshell ~]$
```

sudo dhclient eth0

```
! Validate the leased IP address.
[guestshell@guestshell ~]$
```

sudo ifconfig eth0

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
```

inet 10.1.1.12 netmask 255.255.255.0

```
  broadcast 10.1.1.255
        inet6 fe80::5054:ddff:fe85:a0d5  prefixlen 64  scopeid 0x20
        ether 52:54:dd:85:a0:d5  txqueuelen 1000  (Ethernet)
        RX packets 7  bytes 1000 (1000.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 11  bytes 1354 (1.3 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[guestshell@guestshell ~]$
```

exit

```
exit

! You can validate the leased IP address from Cisco IOS XE too.
513E.D.02-C9300X-12Y-A-17#
```

guestshell run sudo ifconfig eth0

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
```

```
inet 10.1.1.12  netmask 255.255.255.0

  broadcast 10.1.1.255
        inet6 fe80::5054:ddff:fe85:a0d5  prefixlen 64  scopeid 0x20
        ether 52:54:dd:85:a0:d5  txqueuelen 1000  (Ethernet)
        RX packets 28  bytes 2344 (2.2 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 13  bytes 1494 (1.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Use Cases

Use Case 1: Automatic Saving of Configuration Changes on an SCP Server

In some situations, it is beneficial to automatically save switch configurations to a server every time the write memory command is used. This practice helps maintain a record of changes and allows for configuration rollback if necessary. When choosing a server, both TFTP and SCP can be used, but an SCP server offers an additional layer of security.

The Cisco IOS archive feature provides this functionality. However, a significant drawback is that SCP credentials cannot be concealed in the configuration; the server path is displayed in plain text in both the running and startup configurations.

```
Switch#show running-config | section archive
archive
 path scp://cisco:Cisco!123@10.31.121.224/
 write-memory
```

By using the Guest Shell and Python, it is possible to achieve the same functionality while keeping the credentials hidden. This is done by leveraging environment variables within the Guest Shell to store the actual SCP credentials. As a result, the SCP server credentials are not visible in the running configuration.

In this approach, the running configuration only displays the EEM script, while the Python scripts constructs the copy running-config scp: command with the credentials and passes it to the EEM script to be run.

Follow these steps for this example:

1. Copy the **Python script** to /flash/guest-share directory. This script reads the environment variables SCP_USER and SCP_PASSWORD, and returns the copy startup-config scp: command, so that the EEM script can run it. The script requires the SCP server IP address as an argument. Additionally, the script maintains a log of every time that the command write memory is run in a persistent file located at /flash/guest-share/TAC-write-memory-log.txt. This is the Python script:

```
import sys
import os
import cli
```

```
from datetime import datetime

# Get SCP server from the command-line argument (first argument passed)
scp_server = sys.argv[1]  # Expects the SCP server address as the first argument

# Configure CLI to suppress file prompts (quiet mode)
cli.configure("file prompt quiet")

# Get the current date and time
current_time = datetime.now()

# Format the current time for human-readable output and to use in filenames
formatted_time = current_time.strftime("%Y-%m-%d %H:%M:%S %Z")  # e.g., 2025-03-13 14:30:00 UTC
file_name_time = current_time.strftime("%Y-%m-%d_%H_%M_%S")  # e.g., 2025-03-13_14_30_00

# Retrieve SCP user and password from environment variables securely
scp_user = os.getenv('SCP_USER')  # SCP username from environment
scp_password = os.getenv('SCP_PASSWORD')  # SCP password from environment

# Ensure the credentials are set in the environment, raise error if missing
if not scp_user or not scp_password:
    raise ValueError("SCP user or password not found in environment variables!")

# Construct the SCP command to copy the file, avoiding exposure of sensitive data in print
# WARNING: The password should not be shared openly in logs or outputs.
print(f"copy startup-config scp://{scp_user}:{scp_password}@{scp_server}/config-backup-{file_name_

# Save the event in flash memory (log the write operation)
directory = '/flash/guest-share'  # Directory path where log will be saved
file_name = os.path.join(directory, 'TAC-write-memory-log.txt')  # Full path to log file

# Prepare the log entry with the formatted timestamp
line = f'{formatted_time}: Write memory operation.\n'

# Open the log file in append mode to add the new log entry
with open(file_name, 'a') as file:
    file.write(line)  # Append the log entry to the file
```

In this example, the Python script is copied to the switch using a TFTP server:

<#root>

```
Switch#
```

copy tftp://10.207.204.10/cat9k_scp_command.py flash:/guest-share/cat9k_scp_command.py

```
Accessing tftp://10.207.204.10/cat9k_scp_command.py...
Loading cat9k_scp_command.py from 10.207.204.10 (via GigabitEthernet0/0): !
[OK - 917 bytes]
```

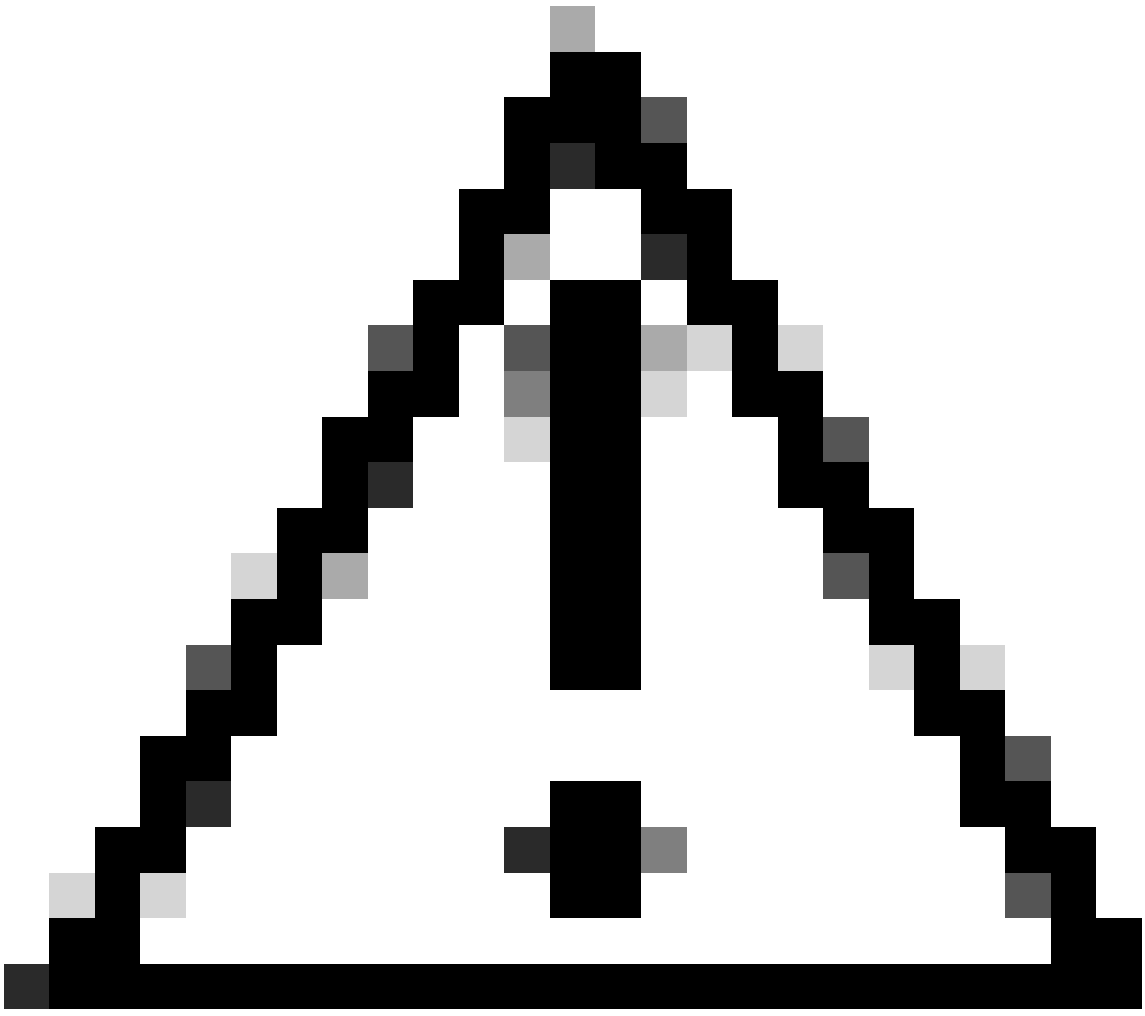
917 bytes copied in 0.017 secs (53941 bytes/sec)

2. Install the **EEM script**. This script calls the Python script. which returns the copy startup-config scp: command needed to save the configuration on the SCP server. The EEM script then runs the command returned by the Python script.

```
event manager applet Python-config-backup authorization bypass
 event cli pattern "^write|write memory|copy running-config startup-config" sync no skip no maxrur
 action 0000 syslog msg "Config save detected, TAC EEM-python started."
 action 0005 cli command "enable"
 action 0015 cli command "guestshell run python3 /bootflash/guest-share/cat9k_scp_command.py 10.31
 action 0020 regexp "(^.*)\n" "$_cli_result" match command
 action 0025 cli command "$command"
 action 0030 syslog msg "TAC EEM-python script finished with result: $_cli_result"
```

3. Set the **Guest Shell environment variables** by inserting them into the ~/.bashrc file. This ensures that the environment variables are persistent every time a Guest Shell is opened, even after the switch is reloaded. Add these two lines:

```
export SCP_USER="cisco"
export SCP_PASSWORD="Cisco!123"
```



> **Caution**: The credentials used in this example are for educational purposes only. They are not intended for use in production environments. Users are required to replace these credentials with their own secure, environment-specific credentials.

This is the process to add these variables to the ~/.bashrc file:

<#root>

```
! 1. Enter a Guest Shell bash session.
Switch#

guestshell run bash



! 2. Locate the ~/.bashrc file.
[guestshell@guestshell ~]$

ls ~/.bashrc

/home/guestshell/.bashrc

! 3. Add the SCP_USER and SCP_PASSWORD environment variables at the end of the ~/.bashrc file.
[guestshell@guestshell ~]$

echo 'export SCP_USER="cisco"' >> ~/.bashrc


[guestshell@guestshell ~]$

echo 'export SCP_PASSWORD="Cisco!123"' >> ~/.bashrc



! 4. To validate these 2 new lines were added correctly, display the content of the ~/.bashrc file
[guestshell@guestshell ~]$

cat ~/.bashrc

# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# User specific environment
if ! [[ "$PATH" =~ "$HOME/.local/bin:$HOME/bin:" ]]
then
    PATH="$HOME/.local/bin:$HOME/bin:$PATH"
fi
export PATH

# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions
[guestshell@guestshell ~]$ echo 'export SCP_USER="cisco"' >> ~/.bashrc
[guestshell@guestshell ~]$ echo 'export SCP_PASSWORD="Cisco!123"' >> ~/.bashrc
[guestshell@guestshell ~]$ cat ~/.bashrc
# .bashrc
```

```
# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# User specific environment
if ! [[ "$PATH" =~ "$HOME/.local/bin:$HOME/bin:" ]]
then
    PATH="$HOME/.local/bin:$HOME/bin:$PATH"
fi
export PATH

# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions

export SCP_USER="cisco"
export SCP_PASSWORD="Cisco!123"

! 5. Reload the ~/.bashrc file in the current session.
[guestshell@guestshell ~]$

source ~/.bashrc



! 6. Validate that the environment variables are added, then exit the Guest Shell session.
[guestshell@guestshell ~]$

printenv | grep SCP
SCP_USER=cisco
SCP_PASSWORD=Cisco!123

[guestshell@guestshell ~]$ exit

Switch#
```

4. Run the **Python script** manually to validate that it returns the right copy command and logs the operation in the persistent TAC-write-memory-log.txt file.

```
<#root>

Switch#

guestshell run python3 /flash/guest-share/cat9k_scp_command.py 10.31.121.224
copy startup-config scp://cisco:Cisco!123@10.31.121.224/config-backup-2025-01-25_18_35_18.txt


Switch#

dir flash:guest-share/

Directory of flash:guest-share/

286725  -rw-              368  Jan 25 2025 18:35:18 +00:00

TAC-write-memory-log.txt

286726  -rw-              903  Jan 25 2025 18:34:45 +00:00  cat9k_scp_command.py
286723  -rw-              144  Jan 25 2025 15:07:07 +00:00  TAC-shutdown-log.txt
```

```
286722  -rw-              977  Jan 25 2025 14:50:56 +00:00  cat9k_noshut.py

11353194496 bytes total (3751542784 bytes free)

Switch#
```

more flash:/guest-share/TAC-write-memory-log.txt
2025-01-25 18:35:18 : Write memory operation.

5. Test the **EEM script**. This EEM script also sends a syslog with the result of the copy operation, whether it is successful or failed. Here is an example of a successful run:

<#root>

Switch#

write memory

Building configuration...
[OK]
Switch#
*Jan 25 19:23:22.189: %HA_EM-6-LOG: Python-config-backup: Config save detected, TAC EEM-python sta
*Jan 25 19:23:42.885: %HA_EM-6-LOG: Python-config-backup:

TAC EEM-python script finished with result:
Writing config-backup-2025-01-25_19_23_26.txt !
8746 bytes copied in 15.175 secs (576 bytes/sec)

Switch#

Switch#

more flash:guest-share/TAC-write-memory-log.txt
2025-01-25 19:23:26 : Write memory operation.

To test a failed transfer, the SCP server is shut down. This is the result of this failed run:

<#root>

Switch#

write

Building configuration...
[OK]
Switch#
*Jan 25 19:25:31.439: %HA_EM-6-LOG: Python-config-backup: Config save detected, TAC EEM-python sta
*Jan 25 19:26:06.934: %HA_EM-6-LOG: Python-config-backup:

TAC EEM-python script finished with result:
Writing config-backup-2025-01-25_19_25_36.txt % Connection timed out; remote host not responding

%Error writing scp://*:*@10.31.121.224/config-backup-2025-01-25_19_25_36.txt (Undefined error)

Switch#
Switch#

```
Switch#
Switch#
more flash:guest-share/TAC-write-memory-log.txt

2025-01-25 19:23:26 : Write memory operation.

2025-01-25 19:25:36 : Write memory operation.
```

Use Case 2: Monitor Increments in STP Topology Changes

This example is useful for monitoring issues related to Spanning Tree instability and identifying which interface is receiving Topology Change Notifications (TCNs). The EEM script is run periodically at a specified time interval and calls a Python script that runs a show command and checks if the TCNs have increased.

Creating this script using only EEM commands would require using for loops and multiple regex matches, which would be cumbersome. Therefore, this example shows how the EEM script delegates this complex logic to Python.

Follow these steps for this example:

1. Copy the **Python script** to /flash/guest-share/ directory. This script performs these tasks:

    1. It runs the show spanning-tree detail command and parses the output to save TCN information for each VLAN in a dictionary.

    2. It compares the parsed TCN information with the data in the JSON file from the previous script run. For each VLAN, if the TCNs have increased, a syslog message is sent with information similar to this example:

    ```
    *Jan 31 18:57:37.852: %GUESTSHELL-5-PYTHON_SCRIPT: Message from tty73(user id: shxUnknownTTY
    ```

    3. It saves the current TCN information in a JSON file to be compared during the next run. This is the Python script:

    ```
    import os
    import json
    import cli
    import re
    from datetime import datetime


    def main():
        # Get TCNs by running the CLI command to show spanning tree details
        tcns = cli.cli("show spanning-tree detail")

        # Parse the output into a dictionary of VLAN details
        parsed_tcns = parse_stp_detail(tcns)

        # Path to the JSON file where VLAN TCN data will be stored
    ```

```
    file_path = '/flash/guest-share/tcns.json'

    # Initialize an empty dictionary to hold stored TCN data
    stored_tcn = {}

    # Check if the file exists and process it if it does
    if os.path.exists(file_path):
        try:
            # Open the JSON file and parse its contents into stored_tcn
            with open(file_path, 'r') as f:
                stored_tcn = json.load(f)
                result = compare_tcn_sets(stored_tcn, parsed_tcns)

                # Check each VLAN in the result and log changes if TCN increased
                for vlan_id, vlan_data in result.items():
                    if vlan_data['tcn_increased']:
                        log_message = (
                            f"TCNs increased in VLAN {vlan_id} "
                            f"from {vlan_data['old_tcn']} to {vlan_data['new_tcn']}. "
                            f"Last TCN seen on {vlan_data['source_interface']}."
                        )
                        # Send log message using CLI
                        cli.cli(f"send log facility GUESTSHELL severity 5 mnemonics PYTHON_S

        except json.JSONDecodeError:
            print("Error: The file contains invalid JSON.")
        except Exception as e:
            print(f"An error occurred: {e}")

    # Write the current TCN data to the JSON file for future comparison
    with open(file_path, 'w') as f:
        json.dump(parsed_tcns, f, indent=4)


def parse_stp_detail(cli_output: str):
    """
    Parses the output of "show spanning-tree detail" into a dictionary of VLANs and their TC

    Args:
        cli_output (str): The raw output from the "show spanning-tree detail" command.

    Returns:
        dict: A dictionary where the keys are VLAN IDs and the values contain TCN details.
    """
    vlan_info = {}

    # Regular expressions to match various parts of the "show spanning-tree detail" output
    vlan_pattern = re.compile(r'^\s*(VLAN|MST)(\d+)\s*', re.MULTILINE)
    tcn_pattern = re.compile(r'^\s*Number of topology changes (\d+)\s*', re.MULTILINE)
    last_tcn_pattern = re.compile(r'last change occurred (\d+:\d+:\d+) ago\s*', re.MULTILINE
    last_tcn_days_pattern = re.compile(r'last change occurred (\d+d\d+h) ago\s*', re.MULTILI
    tcn_interface_pattern = re.compile(r'from ([a-zA-Z]+[\d+\/]+\d+)\s*', re.MULTILINE)

    # Find all VLAN blocks in the output
    vlan_blocks = vlan_pattern.split(cli_output)[1:]
    vlan_blocks = [item for item in vlan_blocks if item not in ["VLAN", "MST"]]

    for i in range(0, len(vlan_blocks), 2):
        vlan_id = vlan_blocks[i].strip()

        # Match the relevant patterns for TCN and related details
        tcn_match = tcn_pattern.search(vlan_blocks[i + 1])
```

```
            last_tcn_match = last_tcn_pattern.search(vlan_blocks[i + 1])
            last_tcn_days_match = last_tcn_days_pattern.search(vlan_blocks[i + 1])
            tcn_interface_match = tcn_interface_pattern.search(vlan_blocks[i + 1])

            # Parse the TCN details and add to the dictionary
            if last_tcn_match:
                tcn = int(tcn_match.group(1))
                last_tcn = last_tcn_match.group(1)
                source_interface = tcn_interface_match.group(1) if tcn_interface_match else None
                vlan_info[vlan_id] = {
                    "id_int": int(vlan_id),
                    "tcn": tcn,
                    "last_tcn": last_tcn,
                    "source_interface": source_interface,
                    "tcn_in_last_day": True
                }
            elif last_tcn_days_match:
                tcn = int(tcn_match.group(1))
                last_tcn = last_tcn_days_match.group(1)
                source_interface = tcn_interface_match.group(1) if tcn_interface_match else None
                vlan_info[vlan_id] = {
                    "id_int": int(vlan_id),
                    "tcn": tcn,
                    "last_tcn": last_tcn,
                    "source_interface": source_interface,
                    "tcn_in_last_day": False
                }

        return vlan_info


    def compare_tcn_sets(set1, set2):
        """
        Compares two sets of VLAN TCN data to determine if TCN values have increased.

        Args:
            set1 (dict): The first set of VLAN TCN data.
            set2 (dict): The second set of VLAN TCN data.

        Returns:
            dict: A dictionary indicating whether the TCN has increased for each VLAN.
        """
        tcn_changes = {}

        # Compare TCN values for VLANs that exist in both sets
        for vlan_id, vlan_data_1 in set1.items():
            if vlan_id in set2:
                vlan_data_2 = set2[vlan_id]
                tcn_increased = vlan_data_2['tcn'] > vlan_data_1['tcn']
                tcn_changes[vlan_id] = {
                    'tcn_increased': tcn_increased,
                    'old_tcn': vlan_data_1['tcn'],
                    'new_tcn': vlan_data_2['tcn'],
                    'source_interface': vlan_data_2['source_interface']
                }
            else:
                tcn_changes[vlan_id] = {
                    'tcn_increased': None,  # No comparison if VLAN is not in set2
                    'old_tcn': vlan_data_1['tcn'],
                    'new_tcn': None
                }
```

```
    # Check for VLANs in set2 that are not in set1
    for vlan_id, vlan_data_2 in set2.items():
        if vlan_id not in set1:
            tcn_changes[vlan_id] = {
                'tcn_increased': None,  # No comparison if VLAN is not in set1
                'old_tcn': None,
                'new_tcn': vlan_data_2['tcn']
            }

    return tcn_changes


if __name__ == "__main__":
    main()
```

In this example, the Python script is copied to the switch using a TFTP server:

<#root>

Switch#

copy tftp://10.207.204.10/cat9k_tcn.py flash:/guest-share/cat9k_tcn.py

Accessing tftp://10.207.204.10/cat9k_tcn.py...
Loading cat9k_tcn.py from 10.207.204.10 (via GigabitEthernet0/0): !
[OK - 5739 bytes]

5739 bytes copied in 0.023 secs (249522 bytes/sec)

2. Install the **EEM script**. In this example, the sole task of the EEM script is to run the Python script, which sends a log message if a TCN increment is detected. In this example, the EEM script runs every 5 minutes.

```
event manager applet tcn_monitor authorization bypass
 event timer watchdog time 300
 action 0000 syslog msg "TAC EEM-python script started."
 action 0005 cli command "enable"
 action 0015 cli command "guestshell run python3 /bootflash/guest-share/cat9k_tcn.py"
 action 0020 syslog msg "TAC EEM-python script finished."
```

3. To validate the functionality of the script, you can either run the **Python script** manually or wait five minutes for the EEM script to call it. In both cases, a syslog is sent only if the TCNs have increased for a VLAN.

<#root>

Switch#

more flash:/guest-share/tcns.json

```
{
"0001": {
"id_int": 1,
"tcn": 20,
"last_tcn": "00:01:18",
"source_interface": "TwentyFiveGigE1/0/5",
"tcn_in_last_day": true
},
"0010": {
"id_int": 10,
"tcn": 2,
"last_tcn": "00:00:22",
"source_interface": "TwentyFiveGigE1/0/1",
"tcn_in_last_day": true
},
"0020": {
"id_int": 20,
"tcn": 2,
"last_tcn": "00:01:07",
"source_interface": "TwentyFiveGigE1/0/2",
"tcn_in_last_day": true
},
"0030": {
"id_int": 30,
"tcn": 1,
"last_tcn": "00:01:18",
"source_interface": "TwentyFiveGigE1/0/3",
"tcn_in_last_day": true
}
}
```


```
Switch#
guestshell run python3 /flash/guest-share/cat9k_tcn.py

Switch#
*Feb 17 21:34:45.846: %GUESTSHELL-5-PYTHON_SCRIPT: Message from tty73(user id: shxUnknownTTY): TCN
Switch#
```

4. Test the **EEM script** by waiting for it to run every five minutes. If the TCNs have increased for any VLAN, a syslog message is sent. In this particular example, it is noted that TCNs are constantly increasing on VLAN 30, and the interface receiving these constant TCNs is Twe1/0/3.

```
<#root>
*Feb 17 21:56:23.563: %HA_EM-6-LOG: tcn_monitor: TAC EEM-python script started.
*Feb 17 21:56:26.039: %GUESTSHELL-5-PYTHON_SCRIPT: Message from tty73(user id: shxUnknownTTY):
TCNs increased in VLAN 0030 from 3 to 5. Last TCN seen on TwentyFiveGigE1/0/3.
```

```
*Feb 17 21:56:26.585: %HA_EM-6-LOG: tcn_monitor: TAC EEM-python script finished.
*Feb 17 22:01:23.563: %HA_EM-6-LOG: tcn_monitor: TAC EEM-python script started.
*Feb 17 22:01:26.687: %HA_EM-6-LOG: tcn_monitor: TAC EEM-python script finished.
*Feb 17 22:06:23.564: %HA_EM-6-LOG: tcn_monitor: TAC EEM-python script started.
*Feb 17 22:06:26.200: %GUESTSHELL-5-PYTHON_SCRIPT: Message from tty73(user id: shxUnknownTTY):

TCNs increased in VLAN 0030 from 5 to 9. Last TCN seen on TwentyFiveGigE1/0/3.

*Feb 17 22:06:26.787: %HA_EM-6-LOG: tcn_monitor: TAC EEM-python script finished.
*Feb 17 22:11:23.564: %HA_EM-6-LOG: tcn_monitor: TAC EEM-python script started.
*Feb 17 22:11:26.079: %GUESTSHELL-5-PYTHON_SCRIPT: Message from tty73(user id: shxUnknownTTY):

TCNs increased in VLAN 0030 from 9 to 12. Last TCN seen on TwentyFiveGigE1/0/3.

*Feb 17 22:11:26.686: %HA_EM-6-LOG: tcn_monitor: TAC EEM-python script finished.
```

Related Information

- Programmability Configuration Guide, Cisco IOS XE Cupertino 17.9.x (Chapter: Guest Shell)
- Understand Best Practices and Useful Scripts for EEM
- Application Hosting on the Cisco Catalyst 9000 Series Switches White paper