使用Python脚本实现Catalyst 9000交换机的自动化

目录

简介

本文档介绍如何使用Python脚本扩展EEM，以在Catalyst 9000交换机上自动执行配置和数据收集。

先决条件

要求

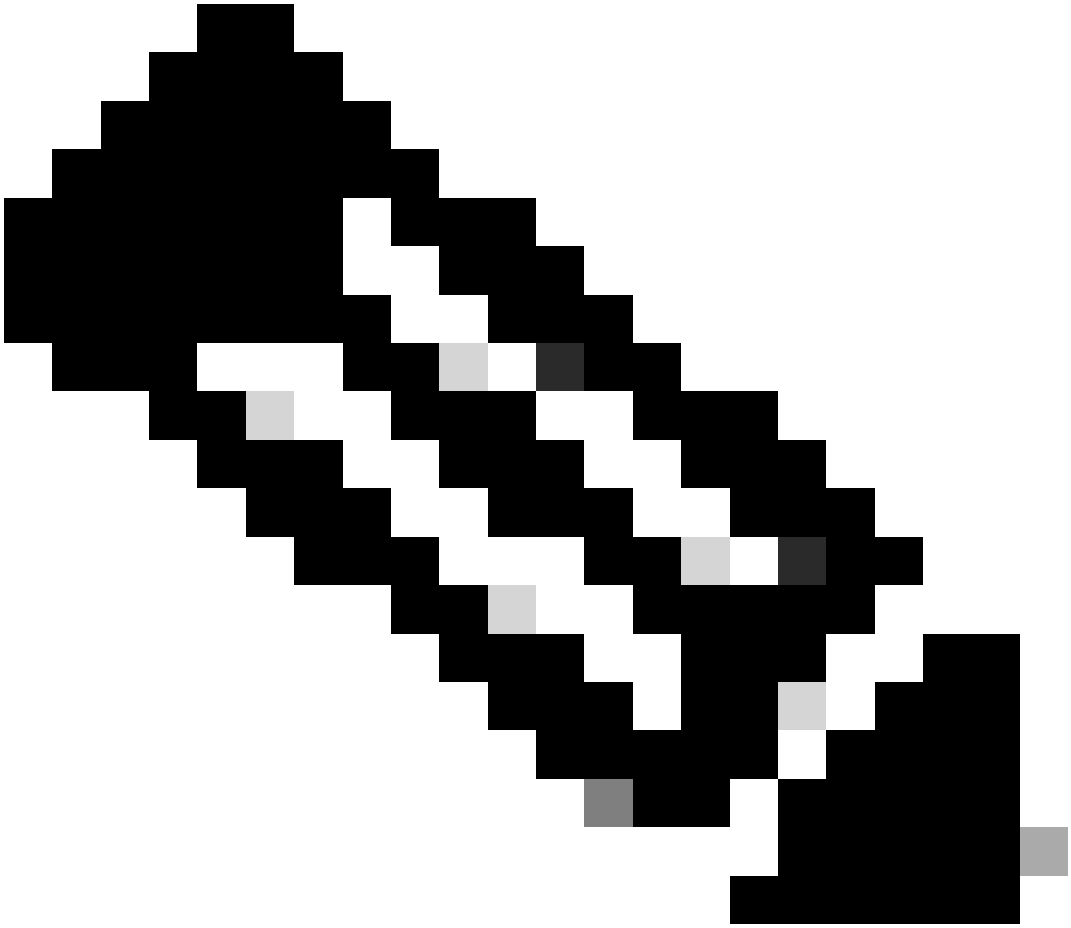
思科建议您了解并熟悉以下主题：

- Cisco IOS®和Cisco IOS® XE EEM
- 应用托管和访客外壳
- Python 脚本
- Linux命令

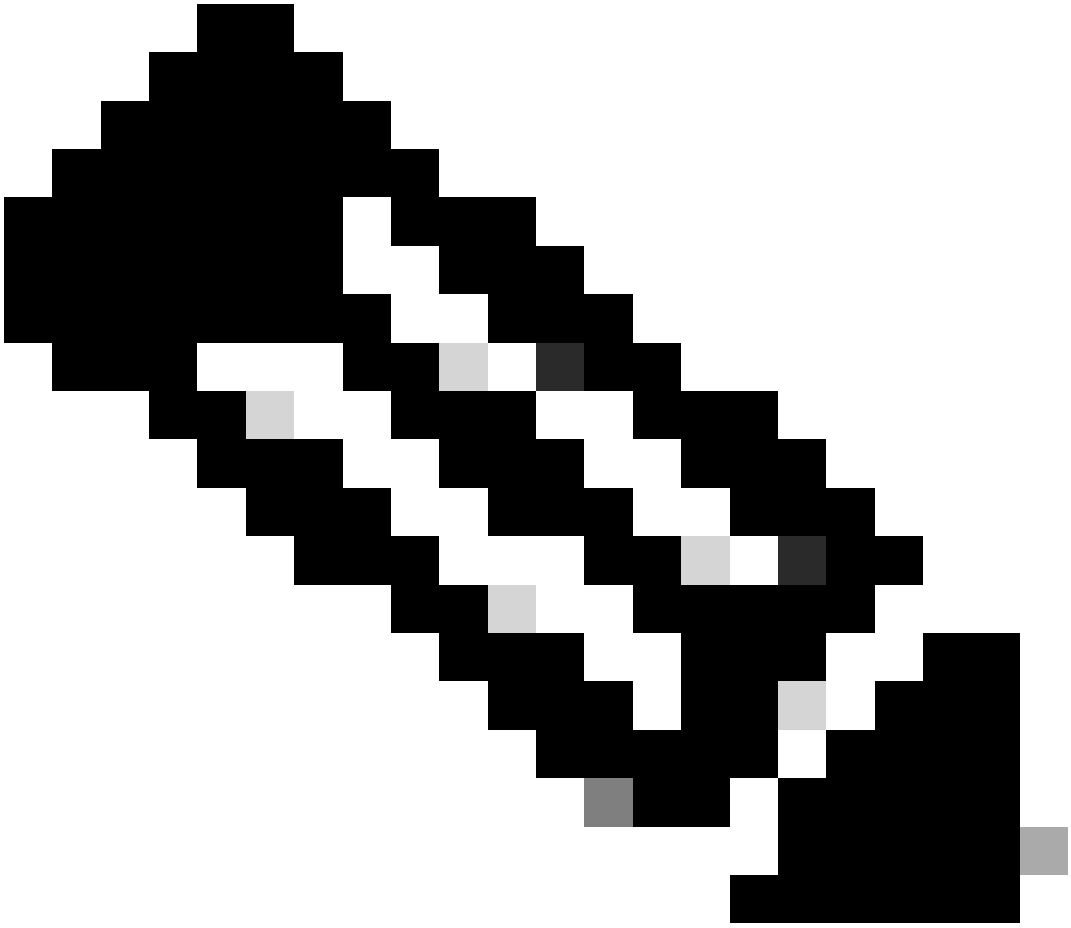
使用的组件

本文档中的信息基于以下软件和硬件版本：

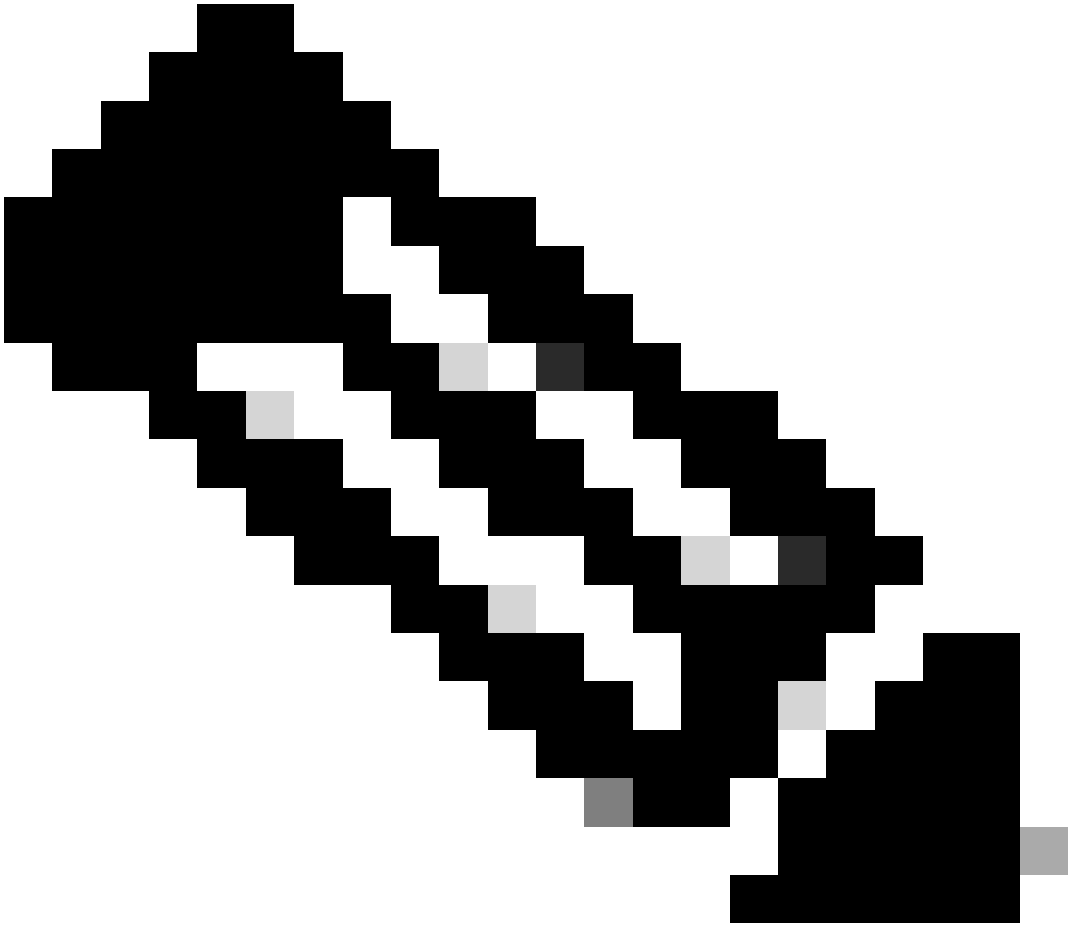
- Catalyst 9200
- Catalyst 9300
- Catalyst 9400
- Catalyst 9500
- Catalyst 9600
- Cisco IOS XE 17.9.1及更高版本



注意：有关在其他思科平台上启用这些功能所使用的命令，请参阅相应的配置指南。



注意：Catalyst 9200L交换机不支持Guest Shell。



注意：思科TAC不支持这些脚本，这些脚本按原样提供，用于教学目的。

本文档中的信息都是基于特定实验室环境中的设备编写的。本文档中使用的所有设备最初均采用原始（默认）配置。如果您的网络处于活动状态，请确保您了解所有命令的潜在影响。

规则

有关文档规则的信息，请参阅 Cisco 技术提示规则。

背景信息

访客Shell和Python脚本

Cisco Catalyst 9000系列交换机上的应用托管为合作伙伴和开发人员带来了创新机会，因为网络设备可以与应用运行时环境合并。

它支持容器化应用，与主操作系统和思科IOS XE内核完全隔离。这种分离可确保托管应用的资源分

配与核心路由和交换功能不同。

Cisco IOS XE设备的应用托管基础设施称为IOx(Cisco IOS + Linux)，便于在网络设备上托管由思科、合作伙伴和第三方开发人员开发的应用和服务，确保跨各种硬件平台无缝集成。

Guest Shell是一种专用容器部署，它是对系统部署有帮助的应用示例。

Guest Shell提供基于Linux的虚拟化环境，旨在运行自定义Linux应用（包括Python），以实现思科设备的自动化控制和管理。Guest Shell容器允许用户在系统中运行脚本和应用。具体来说，在Intel x86平台上，Guest Shell容器是一个Linux容器(LXC)，其中包含CentOS 8.0最小根文件系统。在Cisco IOS XE Amsterdam 17.3.1及更高版本中，仅支持Python V3.6。可以在运行时使用CentOS 8.0中的Yum实用程序安装其他Python库。也可以使用Pip安装包(PIP)安装或更新Python包。

Guest Shell包含Python应用编程接口(API)，允许使用Python CLI模块运行Cisco IOS XE命令。这样，Python脚本增强了自动化功能，为网络工程师提供了一款通用工具，用于开发用于自动化配置和数据收集任务的脚本。虽然这些脚本可以通过CLI手动运行，但是也可以与EEM脚本一起用于响应特定事件，例如系统日志消息、接口事件或命令运行。实际上，任何能够触发EEM脚本的事件也可用于触发Python脚本，从而扩大Cisco Catalyst 9000交换机内部的自动化潜力。

安装Guest Shell后，将在闪存文件系统中自动创建访客共享目录。这是可以通过Python脚本和访客外壳访问的文件系统。为确保使用堆叠时正确同步，请将此文件夹保持在50 MB以下。

将Python与EEM脚本配合使用的优势

- Python允许在Python脚本中处理复杂的逻辑（如正则表达式、循环和匹配），从而扩展了EEM脚本的自动化功能。此功能提供了创建更强大的EEM脚本的机会。
- 作为一种著名的编程语言，Python可降低希望自动化Cisco IOS XE设备的网络工程师的进入障碍。此外，它还提供可维护性和可读性。
- Python还提供错误处理功能以及强大的标准库。

将Python与EEM脚本配合使用的注意事项

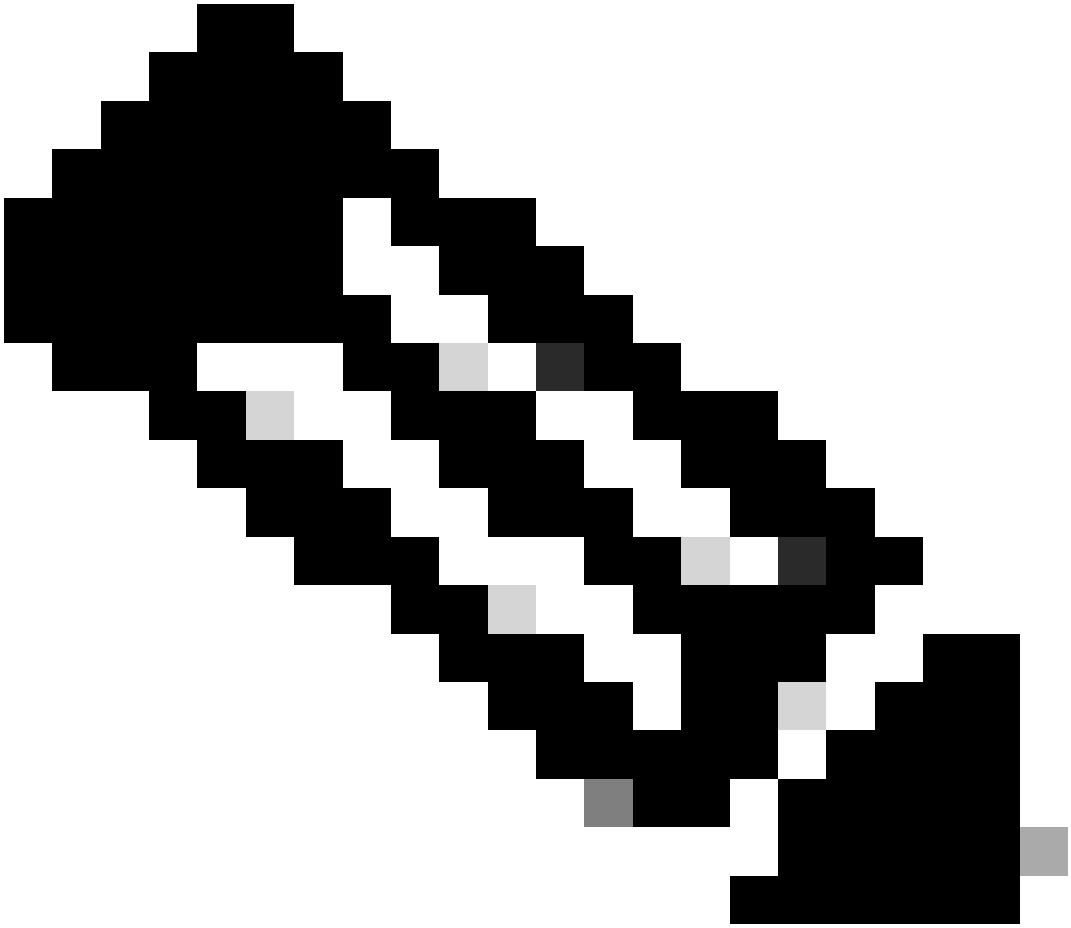
- 默认情况下，Guest Shell未启用，因此需要先启用它，然后才能运行Python脚本。
- 无法直接在CLI中创建Python脚本；它们需要首先在开发环境中开发，然后复制到交换机的闪存中。

Cisco IOS XE中的SELinux

从Cisco IOS XE 17.8.1开始，引入了对安全增强型Linux(SELinux)的支持，以通过管理进程、用户和文件相互交互方式的策略实施安全性。SELinux策略定义允许进程或用户访问哪些操作和资源。当用户或进程尝试执行策略不允许的操作（例如，访问资源或运行命令）时，可能会发生违规。SELinux可以在两种不同的模式下运行：

1. 许可模式：SELinux不会实施任何策略。但是，它仍会记录任何违规行为，好像这些行为被拒绝一样。

2. 实施：SELinux会在系统上主动实施安全策略。如果操作违反SELinux策略，则拒绝并记录操作。



注意：在Cisco IOS XE 17.8.1中引入默认模式时，默认模式被设置为允许模式。但是，从版本17.14.1开始，SELinux将在实施模式下启用。

使用Guest Shell时，在使用实施模式时，可能会拒绝访问某些资源。如果尝试使用Guest Shell或Python脚本执行操作时导致权限被拒绝错误，则与此日志类似：

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

要验证SELinux是否拒绝脚本，请使用命令show platform software audit summary检查拒绝计数是否增加。此外show platform software audit all，还会显示SELinux阻止的操作的日志。访问矢量缓存(AVC)是在SELinux中记录访问控制决策的机制，因此使用此命令时，请查找以type=AVC开头的日志。

如果脚本被拒绝和阻塞，可以使用命令将SELinux设置为允许模式set platform software selinux permissive。此更改不会存储在运行配置或启动配置中，因此重新加载后，模式将恢复为实施。因此，每次重新加载交换机时，都需要重新应用此更改。可使用验证更改show platform software selinux。

配置

要使用EEM和Python脚本自动执行交换机上的任务，请执行以下步骤：

1. 启用Guest Shell。
2. 将Python脚本复制到/flash/guest-share/目录。您可以使用Cisco IOS XE中可用的任何复制机制，例如SCP、FTP或WebUI中的FIle Manager。Python脚本在闪存中后，可以使用命令运行该脚本guestshell run python3 /flash/guest-share/cat9k_script.py。
3. 配置运行Python脚本的EEM脚本。此设置允许使用EEM脚本提供的任意多个事件检测器（例如系统日志消息、CLI模式和Cron调度程序）触发Python脚本。

本节介绍步骤1。下一部分提供了演示如何实施步骤2和步骤3的示例。

使用静态IP地址启用Guest Shell

按照以下过程启用Guest Shell:

1. 启用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. 为Guest Shell配置Application Hosting Network。本示例使用AppGigabitEthernet接口提供网络访问；但是，也可以使用管理接口(Gi0/0)。

```
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. 启用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. 验证Guest Shell。此示例验证默认网关和cisco.com均可访问。此外，验证Python 3是否可以从访客外壳运行。

<#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#
```

使用DHCP IP地址启用Guest Shell

通常，Guest Shell配置有静态IP地址，因为Guest Shell容器默认情况下没有DHCP客户端服务。如果访客外壳需要动态请求IP地址，则需要安装DHCP客户端服务。遵循以下流程：

1. 按照以下步骤使用静态IP地址启用Guest Shell。但是，这次不要在步骤2中分配应用托管配置中的IP地址。请改用以下配置：

```
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. DHCP客户端可以使用带有命令的Yum实用程序进行安装sudo yum install dhcp-client。但是，CentOS Stream 8的存储库已停用。要解决此问题，可以手动下载和安装DHCP客户端软件包。在PC上，从CentOS Stream 8存储区下载这些软件包，并将其打包为tar文件。

   - 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. 将文件复制到交换机的/flash/guest-share/目录dhcp-client.tar。

4. 输入Guest Shell bash session并运行Linux命令以安装DHCP客户端并请求IP地址。

```
<#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
```

使用案例

使用案例1:自动保存SCP服务器上的配置更改

在某些情况下，每次使用命令时自动将交换机配置保存到服务器write memory是有益的。这种做法有助于维护更改记录，并在必要时允许配置回滚。选择服务器时，既可以使用TFTP也可以使用SCP，但SCP服务器提供额外的安全层。

Cisco IOS存档功能提供此功能。但是，SCP凭证在配置中无法隐藏，这是一个重大缺陷；服务器路径在运行配置和启动配置中均以明文显示。

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

通过使用Guest Shell和Python，可以在保持凭证隐藏的同时实现相同的功能。这通过利用Guest Shell中的环境变量来存储实际SCP凭证来实现。因此，SCP服务器凭证在运行配置中不可见。

在此方法中，运行配置仅显示EEM脚本，而Python脚本使用凭证构建命令copy running-config scp:，并将其传递给要运行的EEM脚本。

在本示例中，请按照以下步骤操作：

1. 将Python脚本复制到/flash/guest-share目录。此脚本读取环境变量SCP_USER和SCP_PASSWORD，并返回命令copy startup-config scp:，以便EEM脚本可以运行它。脚本需要将SCP服务器IP地址作为参数。此外，脚本会维护每次在位于/flash/guest-share/TAC-write-memory-log.txt的持久文件中运行命令时的日志write memory。这是Python脚本：

```
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
```

在本示例中，使用TFTP服务器将Python脚本复制到交换机：

```
<#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)
```

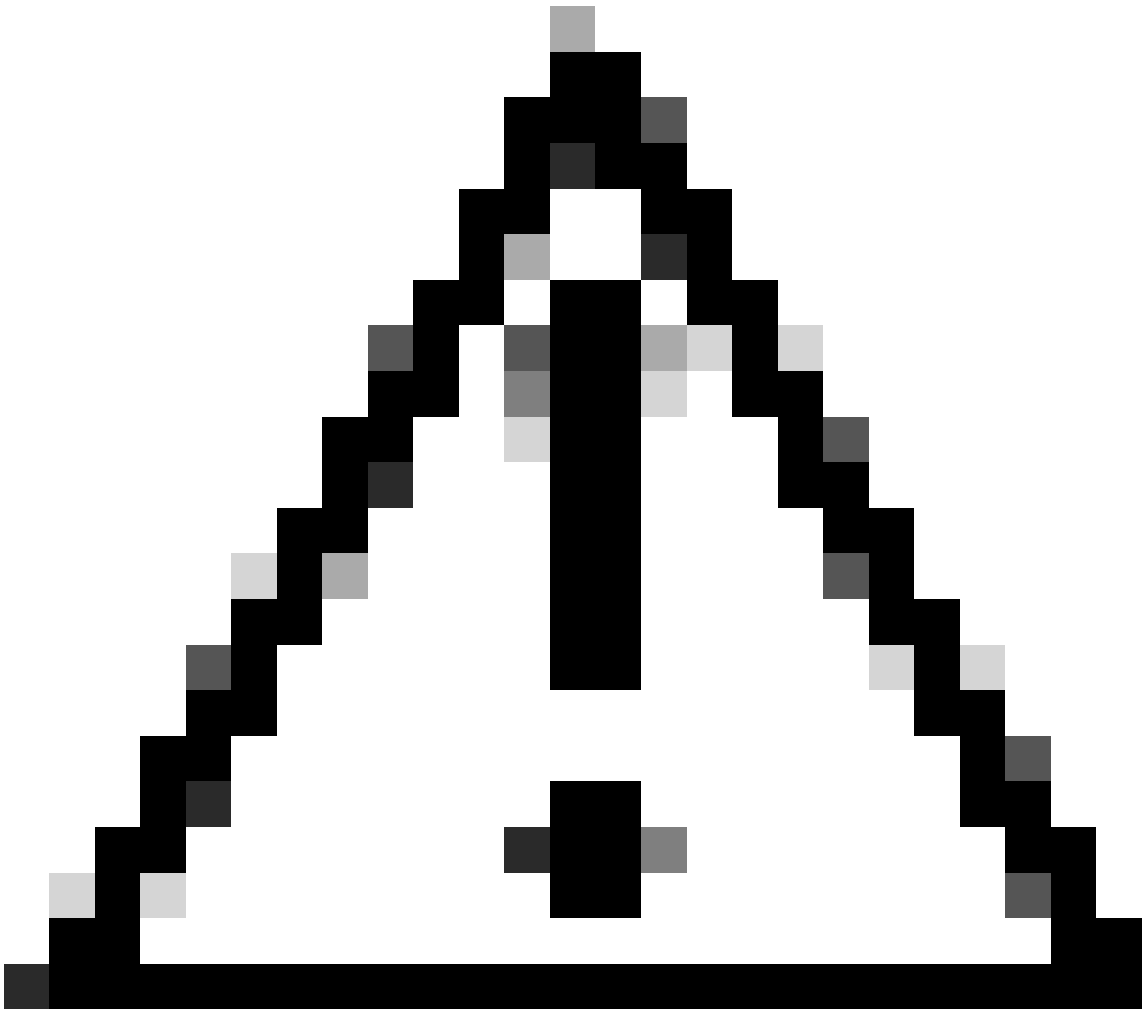
2. 安装EEM脚本。此脚本称为Python脚本。返回在SCP服务器上保存配置所需的命令copy startup-config scp:。然后，EEM脚本运行Python脚本返回的命令。

```
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. 将Guest Shell环境变量插入到文件中，以设置这些变量~/.bashrc。这样可以确保每次打开访客外壳时环境变量都是持久的，即使在重新加载交换机后也是如此。添加以下两行：

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



警告：本示例中使用的凭证仅用于教育目的。它们不能用于生产环境。用户需要用自己的安全环境特定凭证替换这些凭证。

这是向文件添加这些变量的过程~/.bashrc:

<#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. 手动运行Python脚本，以验证它返回正确的命令copy，并将操作记录到永久文件TAC-write-memory-log.txt中。

```
<#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. 测试EEM脚本。此EEM脚本还会发送包含复制操作结果的系统日志，无论复制操作成功还是失败。以下是成功运行的示例：

```
<#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.
```

要测试传输失败，请关闭SCP服务器。这是此失败运行的结果：

```
<#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.
```

使用案例2:监控STP拓扑更改的增量

此示例对于监控与生成树不稳定相关的问题，以及确定哪个接口正在接收拓扑更改通知(TCN)非常有用。 EEM脚本以指定的时间间隔定期运行，并调用运行show命令并检查TCN是否增加的Python脚本。

如果仅使用EEM命令创建此脚本，则需要使用循环和多个正则表达式匹配，这将非常麻烦。因此，此示例展示EEM脚本如何将此复杂逻辑委派给Python。

在本示例中，请按照以下步骤操作：

1. 将Python脚本复制到/flash/guest-share/目录。此脚本执行以下任务：
    1. 它会运行show spanning-tree detail命令并解析输出，以将每个VLAN的TCN信息保存在字典中。
    2. 它将解析的TCN信息与上次运行脚本时的JSON文件中的数据进行比较。对于每个VLAN，如果TCN增加，系统将会发送一条系统日志消息，其中包含类似于以下示例的信息：

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

    3. 它将当前TCN信息保存在JSON文件中，以便在下次运行期间进行比较。这是Python脚本：

```
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()
```

在本示例中，使用TFTP服务器将Python脚本复制到交换机：

```
<#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. 安装EEM脚本。在本示例中，EEM脚本的唯一任务是运行Python脚本，该脚本在检测到TCN增量时发送日志消息。在本示例中，EEM脚本每5分钟运行一次。

```
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. 要验证脚本的功能，您可以手动运行Python脚本，或者等待5分钟让EEM脚本调用该脚本。在这两种情况下，只有当VLAN的TCN增加时，系统才会发送系统日志。

```
<#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. 通过等待EEM脚本每五分钟运行一次来测试该脚本。如果任何VLAN的TCN增加，系统将会发送系统日志消息。在本例中，我们注意到VLAN 30上的TCN不断增加，而接收这些固定TCN的接口是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.
```

相关信息

- 可编程性配置指南，Cisco IOS XE Cupertino 17.9.x(章节：访客外壳)
- 了解EEM的最佳实践和有用的脚本
- Cisco Catalyst 9000系列交换机上的应用托管白皮书

关于此翻译
思科采用人工翻译与机器翻译相结合的方式将此文档翻译成不同语言，希望全球的用户都能通过各自的语言得到支持性的内容。

请注意：即使是最好的机器翻译，其准确度也不及专业翻译人员的水平。

Cisco Systems, Inc. 对于翻译的准确性不承担任何责任，并建议您总是参考英文原始文档（已提供链接）。