Erfassung von IOS-XE Routing Protocol Flapping-Protokollen mit Python

Inhalt

Einleitung

In diesem Dokument wird beschrieben, wie Python-Skripts zum Erfassen von OSPF-, EIGRP- und IS-IS-Protokollen konfiguriert werden, wenn bei den Protokollen Flapping-Ereignisse auftreten.

Voraussetzungen

Anforderungen

Cisco empfiehlt, dass Sie mit den folgenden Themen vertraut sind:

- Konfiguration des Anwendungshostings
- OSPF
- EIGRP
- IS-IS
- vi-Editor

Verwendete Komponenten

Die Informationen in diesem Dokument basieren auf der Cisco IOS XE Software-Version 17.

Die Informationen in diesem Dokument beziehen sich auf Geräte in einer speziell eingerichteten Testumgebung. Alle Geräte, die in diesem Dokument benutzt wurden, begannen mit einer gelöschten (Nichterfüllungs) Konfiguration. Wenn Ihr Netzwerk in Betrieb ist, stellen Sie sicher, dass Sie die möglichen Auswirkungen aller Befehle kennen.

Anmerkung: In diesem Dokument werden die Details zum Apphosting nicht näher erläutert. Weitere Informationen finden Sie in den Links, auf die verwiesen wird.

Konfigurieren

Konfigurationen

Beim Öffnen eines TAC-Tickets ist es sehr wichtig, relevante Informationen zu sammeln, um Zeit zu sparen. Manchmal liegt der Hinweis auf einen Fehler in einigen grundlegenden Ausgaben, die Sie vom Gerät erfassen können. In diesem Dokument finden Sie Beispiele dafür, wie Sie Python-Skripte nutzen können, um diese Daten zu erhalten. Drei Protokolle werden berücksichtigt: OSPF, EIGRP und IS-IS.

Schritt 1. Das erste, was Sie tun müssen, ist, zu konfigurieren und zu aktivieren guestshell.

```
Router(config)#iox
Router(config)#interface VirtualPortGroup 0
Router(config-if)#ip address 192.0.2.1 255.255.255.252
Router(config-if)#exit
Router(config)#
Router(config)#app-hosting appid guestshell
Router(config-app-hosting)#app-vnic gateway0 virtualportgroup 0 guest-interface 0
Router(config-app-hosting-gateway0)#guest-ipaddress 192.0.2.2 netmask 255.255.255.252
Router(config-app-hosting)#app-default-gateway 192.0.2.1 guest-interface 0
Router(config)#end
```

Bei dieser Konfiguration sind drei wichtige Schritte erforderlich:

1. IOX-Dienst aktivieren Dies ist erforderlich, um guestshell zu aktivieren.

2. Konfigurieren Sie die VirtualPortGroup, die als Standardgateway für das Standardgateway der Gastshell fungiert.

3. Konfigurieren Sie das App-Hosting für die Guest Shell. Anhand der Konfigurationen können Sie erkennen, an welcher Stelle die VirtualPortGroup ins Spiel kommt.

Schritt 2. Als Nächstes müssen Sie die Gastshell aus dem privilegierten Modus aktivieren.

```
Router#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

Router#
*Jun 15 21:31:31.499: %IM-6-IOX_INST_INFO: R0/0: ioxman: IOX SERVICE guestshell LOG: Guestshell is up a
```

Wenn alles richtig konfiguriert ist, müssen Sie das Protokoll im vorherigen Beispiel sehen.

Schritt 3: Jetzt können Sie die Python-Skripte konfigurieren. Führen Sie den Befehl guestshell im privilegierten Modus aus. Die Eingabeaufforderung wird wie im folgenden Beispiel angezeigt:

```
Router#guestshell
[guestshell@guestshell ~]$
```

Schritt 4: Erstellen Sie eine Datei mit dem vi-Editor, und konfigurieren Sie die Skripte anhand der aktivierten Protokolle.

```
[guestshell@guestshell ~]$ vi ospf.py
```

Dieses Fenster wird angezeigt.

```
~
~
~
~
~
~
~
"ospf.py" 0L, 0C
```

Schritt 5. Drücken Sie "i", um Text einzufügen. Fügen Sie das Skript ein, drücken Sie "esc", und geben Sie die Zeichen :wq ein.

```
~
from cli import cli
from time import sleep

cli("enable")
cli("debug ip ospf hello")
cli("debug ip ospf adj")
cli("show ip ospf interface | append bootflash:Router-ospf-logs.txt")
cli("show ip ospf neighbor | append bootflash:Router-ospf-logs.txt")
cli("show interfaces | append bootflash:Router-ospf-logs.txt")
cli("show logging | append bootflash:Router-ospf-logs.txt")
cli("show tech | append bootflash:Router-showtech.txt")
sleep(30)
cli("undebug all")
~
~
~
~
"ospf.py" [New] 14L, 458C written
[guestshell@guestshell ~]$
```

Verlassen Sie die guestshell mit dem Befehl exit.

Überprüfung

Testen des Skripts Beenden Sie das Programm mit dem Befehl exit. Führen Sie dann den Befehl guestshell und python3 ospf.py aus.

```
F340.20.09-8500-1#guestshell run python3 ospf.py
```

Hier sehen Sie die Skripte für alle drei Protokolle. OSPF, EIGRP und IS-IS.

OSPF

```
from cli import cli
from time import sleep

cli("enable")
cli("debug ip ospf hello")
cli("debug ip ospf adj")
```

```
cli("show ip ospf interface | append bootflash:Router-ospf-logs.txt")
cli("show ip ospf neighbor | append bootflash:Router-ospf-logs.txt")
cli("show interfaces | append bootflash:Router-ospf-logs.txt")
cli("show logging | append bootflash:Router-ospf-logs.txt")
cli("show tech | append bootflash:Router-showtech.txt")
sleep(30)
cli("undebug all")
```

EIGRP

```
from cli import cli
from time import sleep

cli("enable")
cli("debug eigrp packet")
cli("show ip eigrp neighbor | append bootflash:Router-eigrp-logs.txt")
cli("show ip eigrp interface | append bootflash:Router-eigrp-logs.txt")
cli("show interfaces | append bootflash:Router-eigrp-logs.txt")
cli("show logging | append bootflash:Router-eigrp-logs.txt")
cli("show tech | append bootflash:Router-showtech.txt")
sleep(30)
cli("undebug all")
```

IS-IS

```
from cli import cli
from time import sleep

cli("enable")
cli("debug isis adj-packet")
cli("show isis neighbor detail | append bootflash:Router-isis-logs.txt")
cli("show clns neighbor detail | append bootflash:Router-isis-logs.txt")
cli("show clns interface | append bootflash:Router-isis-logs.txt")
cli("show interfaces | append bootflash:Router-isis-logs.txt")
cli("show logging | append bootflash:Router-isis-logs.txt")
cli("show tech | append bootflash:Router-showtech.txt")
sleep(30)
cli("undebug all")
```

Sie können die Protokollerfassung mit EEM-Skripten automatisieren, die die Python-Skripte ausführen, nachdem Syslog-Muster beobachtet wurden. Im nächsten Abschnitt finden Sie die EEM-Skripte, die Sie zusammen mit den Python-Skripten konfigurieren können, um diese Aufgabe zu erfüllen.

OSPF

```
event manager applet ospf-flap authorization bypass
event syslog pattern "%OSPF-5-ADJCHG:.*from FULL to DOWN" maxrun 120 ratelimit 120
action 010 cli command "enable"
action 020 cli command "guestshell run python3 ospf.py"
action 030 exit
```

EIGRP

```
event manager applet eirgp-flap authorization bypass
event syslog pattern "%DUAL-5-NBRCHANGE: EIGRP.*Neighbor.*is down" maxrun 120 ratelimit 120
action 010 cli command "enable"
action 020 cli command "guestshell run python3 eigrp.py"
action 030 exit
```

IS-IS

```
event manager applet isis-flap authorization bypass
event syslog pattern "%CLNS-5-ADJCHANGE: ISIS: Adjacency to.*Down" maxrun 120 ratelimit 120
action 010 cli command "enable"
action 020 cli command "guestshell run python3 isis.py"
action 030 exit
```

Anmerkung: Die in diesem Skript gesammelten Befehle stellen grundlegende Anfangsinformationen bereit. Wenn Sie ein TAC-Ticket erstellen, können TAC-Techniker weitere Informationen anfordern, um das Ticket bei Bedarf weiter zu untersuchen.

Referenzlinks

- Guestshell
- Python-API

Informationen zu dieser Übersetzung

Cisco hat dieses Dokument maschinell übersetzen und von einem menschlichen Übersetzer editieren und korrigieren lassen, um unseren Benutzern auf der ganzen Welt Support-Inhalte in ihrer eigenen Sprache zu bieten. Bitte beachten Sie, dass selbst die beste maschinelle Übersetzung nicht so genau ist wie eine von einem professionellen Übersetzer angefertigte. Cisco Systems, Inc. übernimmt keine Haftung für die Richtigkeit dieser Übersetzungen und empfiehlt, immer das englische Originaldokument (siehe bereitgestellter Link) heranzuziehen.