Raccolta dei log di flapping del protocollo di routing IOS-XE con Python

Sommario

Introduzione

In questo documento viene descritto come configurare gli script Python per raccogliere i log OSPF, EIGRP e IS-IS quando i protocolli si interrompono.

Prerequisiti

Requisiti

Cisco raccomanda la conoscenza degli argomenti elencati:

- Configurazione hosting app
- OSPF
- EIGRP
- IS-IS
- editor vi

Componenti usati

Il riferimento delle informazioni contenute in questo documento è il software Cisco IOS XE versione 17.

Le informazioni discusse in questo documento fanno riferimento a dispositivi usati in uno specifico ambiente di emulazione. Su tutti i dispositivi menzionati nel documento la configurazione è stata ripristinata ai valori predefiniti. Se la rete è operativa, valutare attentamente eventuali conseguenze derivanti dall'uso dei comandi.

Nota: Questo documento non fornisce i dettagli di hosting. Per ulteriori informazioni, vedere i collegamenti a cui si fa riferimento.

Configurazione

Configurazioni

Quando si apre una richiesta TAC, è molto importante raccogliere informazioni pertinenti per risparmiare tempo. A volte, l'indizio di un guasto si trova all'interno di alcuni output di base che è possibile raccogliere dal dispositivo. In questo documento vengono forniti alcuni esempi di come utilizzare gli script Python per ottenere questi dati. Vengono presi in considerazione tre protocolli, OSPF, EIGRP e IS-IS.

Passaggio 1. La prima cosa da fare è configurare e abilitare 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
```

In questa configurazione, è necessario eseguire tre importanti passaggi:

1. Abilitare il servizio IOX. Questa operazione è necessaria per abilitare guestshell.

2. Configurare il gruppo di porte virtuali che funge da gateway predefinito per il gateway predefinito della shell del guest.

3. Configurare l'hosting dell'app per la shell dei guest. È possibile distinguere dalle configurazioni in cui entra in gioco il gruppo di porte virtuali.

Passaggio 2. È quindi necessario abilitare la shell dei guest dalla modalità privilegiata.

```
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
```

Se tutti gli elementi sono configurati correttamente, è necessario visualizzare il registro nell'esempio precedente.

Passaggio 3. A questo punto, è possibile configurare gli script python. Eseguire il comando guestshell in modalità privilegiata. Il prompt viene visualizzato come nell'esempio seguente:

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

Passaggio 4. Creare un file con l'editor VI e configurare gli script in base ai protocolli abilitati.

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

Viene visualizzata questa finestra

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

Passaggio 5. Premere "i" per inserire il testo. Incollare lo script, premere "esc", quindi immettere i caratteri :wq

```
~
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 ~]$
```

Uscire dalla shell del guest con il comando exit.

Verifica

Testare lo script. Uscire dalla shell del guest con il comando exit. Quindi eseguire guestshell run python3 ospf.py

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

Di seguito sono riportati gli script per tutti e tre i protocolli. OSPF, EIGRP e 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")
```

È possibile automatizzare la raccolta dei log con gli script EEM che eseguono gli script Python dopo aver osservato i pattern syslog. Nella sezione successiva, sono disponibili gli script EEM che è possibile configurare insieme agli script python per eseguire questa operazione.

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
```

Nota: I comandi raccolti in questi script forniscono informazioni iniziali di base. Quando si apre una richiesta TAC, i tecnici possono richiedere ulteriori informazioni per ulteriori informazioni, se necessario.

Link di riferimento

- Guest shell
- API Python

Informazioni su questa traduzione

Cisco ha tradotto questo documento utilizzando una combinazione di tecnologie automatiche e umane per offrire ai nostri utenti in tutto il mondo contenuti di supporto nella propria lingua. Si noti che anche la migliore traduzione automatica non sarà mai accurata come quella fornita da un traduttore professionista. Cisco Systems, Inc. non si assume alcuna responsabilità per l’accuratezza di queste traduzioni e consiglia di consultare sempre il documento originale in inglese (disponibile al link fornito).