Troubleshooting y Prueba de Scripts EEM

Contenido

[Introducción](#)
[Prerequisites](#)
[Requirements](#)
[Componentes Utilizados](#)
[Antecedentes](#)
[Validación de EEM con comandos Show](#)
[Confirmar que los temporizadores están activos](#)
[Confirmar que se están desencadenando eventos desencadenadores](#)
[Revisar historial de eventos](#)
[Validación de EEM con disparador manual](#)
[Consideraciones operativas](#)
[Problema: los comandos CLI no se pueden ejecutar](#)
[Problema: las acciones de EEM tardan más que el tiempo de ejecución máximo](#)
[Problema: EEM se activa con demasiada frecuencia](#)
[Información Relacionada](#)

Introducción

En este documento se describe la validación de secuencias de comandos de Embedded Event Manager (EEM) y se presentan consideraciones operativas y escenarios de fallos comunes.

Prerequisites

Requirements

Este documento asume que el lector ya está familiarizado con la función Embedded Event Manager (EEM) de Cisco IOS/IOS XE. Si todavía no está familiarizado con esta función, lea la Descripción general de la función EEM primero.

EEM en la familia de switches Catalyst 9K requiere el complemento DNA para el nivel de licencia de Network Essentials. Network Advantage es totalmente compatible con EEM.

Componentes Utilizados

La información de este documento se relaciona con la versión 4.0 de EEM implementada en la familia de switches Catalyst.

La información que contiene este documento se creó a partir de los dispositivos en un ambiente de laboratorio específico. Todos los dispositivos que se utilizan en este documento se pusieron en funcionamiento con una configuración verificada (predeterminada). Si tiene una red en vivo, asegúrese de entender el posible impacto de cualquier comando.

Antecedentes

EEM es una función útil cuando se implementa de manera efectiva, pero es importante asegurarse de que

EEM haga exactamente lo que el autor pretende. Los guiones mal revisados pueden provocar problemas catastróficos en la producción. En el mejor de los casos, el script se ejecuta de una manera no deseada. Este documento proporciona información útil sobre cómo probar y verificar EEM con los comandos show de CLI, y también explica algunos escenarios de fallas comunes y las depuraciones utilizadas para identificar y corregir el problema.

Validación de EEM con comandos Show

Confirmar que los temporizadores están activos

Cuando se implementa una secuencia de comandos EEM que es activada por un temporizador, si la secuencia de comandos no se inicia como se esperaba, confirme que el temporizador está activo y cuenta atrás.

Considere estos scripts EEM denominados test y test3 respectivamente:

<#root>

```
event manager
```

applet test

```
 authorization bypass
event timer watchdog time 60
action 0010 syslog msg "Test script running"

event manager
```

applet test3

```
 authorization bypass
event timer watchdog name test3 time 300
action 0010 syslog msg "test3 script running"
```

- El primer script (prueba) utiliza un temporizador de vigilancia de 60 segundos (sin nombre) para activar el script.
- El segundo script (test3) utiliza un temporizador de vigilancia de 300 segundos denominado test3 para activar el script.

Los temporizadores configurados y el valor actual de estos temporizadores se pueden ver con el comando **show event manager statistics server**.

Ejemplo:

<#root>

```
Switch#
```

show event manager statistics server

```
EEM Queue Information
                        Triggered Dropped Queue Queue Average
Client                  Events    Events  Size  Max   Run Time
--------------------------------------------------------------------------------
```

```
Call Home               5         0       0     64    0.021
EEM Applets             181       0       0     64    0.003
EEM IOS .sh Scripts     0         0       0     128   0.000
EEM Tcl Scripts         0         0       0     64    0.000
iosp_global_eem_proc    30        0       0     16    0.004
onep event service init 0         0       0     128   0.000

EEM Policy Counters
Name Value
--------------------------------------------------------------------------------


EEM Policy Timers


Name                      Type

Time Remaining <-- EEM Countdown timer


--------------------------------------------------------------------------------

_EEMinternalname0

         watchdog              53.328

<--- Unnamed timers receive an internal name - this timer is for the 'test' policy


_EEMinternalname1         watchdog              37.120

test3

                     watchdog              183.232

<--- Named timers use their configured name - this is the named timed configured for policy 'test3'
```

Confirmar que se están desencadenando eventos desencadenadores

Como se explicó en la sección Confirmar que los temporizadores están activos de este documento, IOS XE incrementa la columna Eventos desencadenados para la fila de cliente Applets de EEM en la salida del servidor show event manager statistics cada vez que se activa un applet de EEM. Para verificar que la secuencia de comandos de EEM funciona como se espera, ejecute el evento desencadenador varias veces y examine la salida del servidor show event manager statistics para confirmar los incrementos de este valor. Si no es así, la secuencia de comandos no se ha activado.

Cuando el comando se ejecuta varias veces en secuencia, los valores del temporizador deben contarse hacia abajo. Cuando el temporizador llega a cero y el script se ejecuta, el conteo de eventos desencadenados para los Applets EEM también cuenta hacia arriba.

```
<#root>

Switch#

show event manager statistics server


EEM Queue Information
```

```
Triggered

 Dropped Queue Queue Average
Client

Events

    Events  Size  Max   Run Time
-------------------------------------------------------------------------------
Call Home               5         0       0     64    0.021

EEM Applets             183

       0       0     64    0.003

<--- "Triggered Events" column is incremented by 2 due to 2 timers firing


EEM IOS .sh Scripts     0         0       0     128   0.000
EEM Tcl Scripts         0         0       0     64    0.000
iosp_global_eem_proc    30        0       0     16    0.004
onep event service init 0         0       0     128   0.000

EEM Policy Counters
Name Value
-------------------------------------------------------------------------------


EEM Policy Timers
Name                           Type

Time Remaining


-------------------------------------------------------------------------------

_EEMinternalname0

              watchdog        56.215
_EEMinternalname1              watchdog        100.006

test3

                         watchdog        126.117
```

Nota: Si esto no ocurre, investigue el script para verificar los temporizadores configurados.

Revisar historial de eventos

Para scripts que no son activados por temporizadores, el comando show event manager history events es útil para confirmar que los applets se activan como se espera.

Tenga en cuenta este guion de EEM:

```
<#root>

event manager

applet test_manual

 authorization bypass
```

```
event none                                                    <-- manual trigger type for testing

action 0010
syslog msg "I am a manually triggered script!" <-- message that is printed when script runs
```

Esta secuencia de comandos se ejecuta cuando se ejecuta test_manual del administrador de eventos de CLI e imprime un mensaje de syslog. Además de la salida en syslog, la ejecución de este script se puede verificar mediante una revisión de la salida de los eventos show event manager history como se muestra:

```
<#root>

Switch#
show event manager history events

No. Job Id Proc Status   Time of Event
Event Type
           Name
1   5      Actv success  Fri Nov 6 15:45:07 2020
timer countdown

callback: Call Home process  <-- timer bases event that fired

2   18     Actv success  Mon Nov 9 14:12:33 2020   oir                 callback: Call Home process
3   19     Actv success  Mon Nov 9 14:12:40 2020   oir                 callback: Call Home process
4   20     Actv success  Fri Nov13 14:35:49 2020
none

applet: test_manual          <-- manually triggered event
```

Validación de EEM con disparador manual

Hay escenarios en los que es deseable activar manualmente un script EEM, ya sea para probar el flujo de ejecución o para realizar una acción única. Esto se puede lograr con un script EEM con un disparador de evento none como se muestra en esta salida:

```
<#root>

event manager
applet test_manual
 authorization bypass
event none
action 0010 syslog msg "I am a manually triggered script!"
```

Ejecute manualmente la secuencia de comandos con el comando **event manager run test_manual** desde el mensaje de activación:

```
<#root>

Switch#

event manager run test_manual <-- Manually runs the script


Switch#

show log <-- Check for the log from action 10.


*Oct 26 21:24:40.762:

%HA_EM-6-LOG: test_manual: I am a manually triggered script!  <-- %HA_EM logs are from EEM events. The s
```

Consideraciones operativas

Asegúrese de que los scripts EEM se validan antes de su uso en producción. En general, hay algunas formas principales en las que un script no funciona como se esperaba, tres de las cuales se analizan aquí.

Esta sección muestra cómo verificar estos 3 problemas comunes con los scripts EEM:

1. Fallos del comando CLI: el comando no se analiza y, por lo tanto, no se ejecuta.
2. La secuencia de comandos se ejecuta durante demasiado tiempo: las secuencias de comandos de EEM tienen un límite de tiempo de ejecución predeterminado de 20 segundos. Si se supera este tiempo, el script se detiene antes de que se ejecuten todos los comandos.
3. El script se ejecuta con demasiada frecuencia: a veces, el evento desencadenador utilizado por el script puede ocurrir con demasiada frecuencia, lo que hace que el script se desencadene rápidamente. Es deseable controlar con qué frecuencia y a qué velocidad se dispara el script.

Problema: los comandos CLI no se pueden ejecutar

Este ejemplo de script contiene varios problemas. Es un applet simple que agrega la salida de varios comandos show a un archivo de texto en un medio Flash local:

```
<#root>

event manager

applet Data_Collection

 auth bypass
event timer

watchdog time 60


action 1.0 cli command "enable"
action 1.1 cli command "show clock | append flash:DataCollection.txt"
action 1.2 cli command "show interfaces breif | append flash:DataCollection.txt"
action 1.3 cli command "show ip route | append flash:Datacollection.txt"
action 1.4 cli command "show processes cpu sorted | exclude 0.0 | append flash:DataCollection.txt"
```

```
action 1.5 cli command "show platform hardware fed switch active qos stats internal cpu policer | appenc
action 2.0 syslog msg  "Data Capture Complete"
```

El applet se ejecutó correctamente, pero no generó los resultados esperados:

<#root>

```
Switch#
```

show logging | in Capture

<-- Our script-generated syslog contains the string "Capture".

```
*Mar 11 20:40:01.941: %HA_EM-6-LOG: Data_Collection: Data Capture Complete
```

<-- Action 2.0 successfully ran.

```
Switch#
```

dir flash: | in .txt

<-- We only expected one .txt file, however two appear in flash:

```
32792 -rw- 36 Mar 11 2021 20:40:01 +00:00 DataCollection.txt
32798 -rw- 807 Mar 11 2021 20:40:01 +00:00 Datacollection.txt
```

```
Switch#
```

more flash:DataCollection.txt

<-- the output of our expected .txt file is empty except for the output of "show clock

```
"
*20:40:01.343 UTC Thu Mar 11 2021
```

Utilice **debug embedded event manager action cli** para ayudar con la verificación del applet.

<#root>

```
Switch#
```

debug embedded event manager action cli

```
*Mar 11 20:40:01.175: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : CTL : cli_open called.
```

<-- The applet is called.

*Mar 11 20:40:01.275: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : Switch>
*Mar 11 20:40:01.275: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : IN : Switch>enable
*Mar 11 20:40:01.285: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : Switch#
*Mar 11 20:40:01.285: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : IN : Switch#show clock | append
*Mar 11 20:40:01.396: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : Switch#
*Mar 11 20:40:01.396: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : IN : Switch#show interfaces bre
*Mar 11 20:40:01.507: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT :

show interfaces breif

| append flash:DataCollection.txt

<-- Here is our first problem. "brief" is misspelled, so the command does not run.

*Mar 11 20:40:01.507: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT :

^

*Mar 11 20:40:01.507: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT :

% Invalid input detected at '^' marker. <-- CLI parser failure

*Mar 11 20:40:01.507: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT :
*Mar 11 20:40:01.507: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : Switch#
*Mar 11 20:40:01.507: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : IN : Switch#

show ip route | append flash:Datacollection.txt <-- This created the second .txt file. The file name is

***Mar 11 20:40:01.618: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : Switch#**

*Mar 11 20:40:01.618: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : IN : Switch#

show processes cpu sorted | exclude 0.0 | append flash:DataCollection.txt

<-- This problem is less intuitive.

*Mar 11 20:40:01.729: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : CPU utilization for five
*Mar 11 20:40:01.729: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : PID Runtime(ms) Invoked u

the "exclude" argument reads everything beyond the pipe as the value that is to be excluded

.
*Mar 11 20:40:01.729: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 117 57246 448028 127 0.07

A problem like this will likely not be evident in debugging

.
*Mar 11 20:40:01.729: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 2 4488 16816 266 0.07% 0.

This underscores the importance of pre-production testing to ensure the script performs as expected

.
*Mar 11 20:40:01.729: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 173 829 44093 18 0.07% 0.
*Mar 11 20:40:01.729: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 205 22271 1313739 16 0.07
*Mar 11 20:40:01.729: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 467 238 2238 106 0.07% 0.

*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 81 12793 151345 84 0.07%
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 232 22894 2621198 8 0.07%

```
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 7 0 1 0 0.00% 0.00% 0.00%
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 6 0 1 0 0.00% 0.00% 0.00%
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 8 17 2804 6 0.00% 0.00% 0
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 9 33511 11402 2939 0.00%
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 12 0 2 0 0.00% 0.00% 0.00
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 10 106 1402 75 0.00% 0.00
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 14 439 42047 10 0.00% 0.0
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 11 0 1 0 0.00% 0.00% 0.00
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 16 0 1 0 0.00% 0.00% 0.00

*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 5 0 1 0 0.00% 0.00% 0.00%
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 18 0 3 0 0.00% 0.00% 0.00
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : CTL : 20+ lines read from cli,

*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : IN : Switch#
show platform hardware fed switch active qos stats internal cpu policer
 | append flash:DataCollection.txt
*Mar 11 20:40:01.941: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : show platform hardware fe
<-- Here, the syntax of the command was not properly parsed out before implementation. We are missing an


*Mar 11 20:40:01.941: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT :
  ^  <-- missing word queue


*Mar 11 20:40:01.941: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT :
% Invalid input detected at '^' marker.        <-- CLI parser failure


*Mar 11 20:40:01.941: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT :
*Mar 11 20:40:01.941: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : Switch#
*Mar 11 20:40:01.941: %HA_EM-6-LOG: Data_Collection: Data Capture Complete
<-- The syslog from Action 2.0 writes.


*Mar 11 20:40:01.941: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : CTL : cli_close called.
<-- The applet closes out as expected after executing all configured actions.
```

Conclusión: Verifique correctamente todas las acciones de EEM y utilice depuraciones para comprobar las configuraciones erróneas y los errores tipográficos.

Problema: las acciones de EEM tardan más que el tiempo de ejecución máximo

En esta situación, se utiliza un EEM simple para recopilar capturas de paquetes del plano de control en intervalos de 120 segundos. Agrega nuevos datos de captura a un archivo de salida ubicado en un medio de almacenamiento local.

<#root>

```
event manager
```

```
applet Capture


event timer

watchdog time 120     <-- 120 second countdown timer


action 1.0 cli command "enable"
action 1.1 cli command "no monitor capture CPUCapture"
action 2.0 cli command "monitor capture CPUCapture control-plane in match any buffer circular"
action 2.1 cli command "monitor capture CPUCapture start"
action 3.0 wait 45
action 4.0 cli command "monitor capture CPUCapture stop"
action 4.1 cli command "show clock | append flash:CPUCapture.txt"
action 4.2 cli command "show mon cap CPUCapture buff dump | append flash:CPUCapture.txt"

action 5.0 syslog msg "CPUCapture Complete - Next capture in 2 minutes"
```

Puede determinar fácilmente que el EEM no se completa como se esperaba. Verifique los registros locales para el syslog de la acción 5.0. Este syslog se imprime en cada iteración correcta del applet. El registro no se imprimió dentro del búfer y el archivo CPUCapture.txt no se escribió en la memoria flash:

```
<#root>

Switch#

show logging | include "CPUCapture Complete"


Switch#

dir flash: | include CPUCapture.txt
```

Habilite los debugs para investigar. La depuración más comúnmente utilizada es la **CLI de acción del administrador de eventos de depuración.** Esta utilidad imprime un diálogo de las acciones en secuencia.

Resultado de depuración: el resultado de la depuración muestra el applet llamado correctamente. Las acciones iniciales se ejecutan sin problemas, pero la captura no puede concluir.

```
<#root>

Switch#

debug event manager action cli


*Jan 28 22:55:54.742: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : CTL : cli_open called.

<-- This is the initial message seen when the applet is called.


*Jan 28 22:55:54.843: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : OUT : CoreSwitch>

The applet name can be seen within the line.
```

```
*Jan 28 22:55:54.843: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : IN : CoreSwitch>enable
*Jan 28 22:55:54.854: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : OUT : CoreSwitch#
*Jan 28 22:55:54.854: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : IN : CoreSwitch#no monitor capture CPUC
*Jan 28 22:55:54.964: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : OUT : Capture does not exist
*Jan 28 22:55:54.964: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : OUT :
*Jan 28 22:55:54.964: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : OUT : CoreSwitch#
*Jan 28 22:55:54.965: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : IN : CoreSwitch#monitor capture CPUCapt
Jan 28 22:55:55.075: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : OUT : CoreSwitch#
*Jan 28 22:55:55.075: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : IN : CoreSwitch#monitor capture CPUCapt
*Jan 28 22:55:55.185: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : OUT : Started capture point : CPUCaptur
```

<-- The applet successfully creates and starts the capture.

```
*Jan 28 22:55:55.185: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : OUT : CoreSwitch#
*Jan 28 22:56:15.187: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : CTL : cli_close called.
```

<-- After 20 seconds, cli_close is called and the applet begins to exit.

```
*Jan 28 22:56:15.187: fh_server: fh_io_ipc_msg: received msg FH_MSG_CALLBACK_DONE from client 27 pclient
*Jan 28 22:56:15.187: fh_io_ipc_msg: EEM callback policy Capture has ended with abnormal exit status of
```

FF

```
*Jan 28 22:56:15.187:
```

EEM policy Capture has exceeded it's elapsed time limit of 20.0 seconds <-- We are informed that the pol

```
*Jan 28 22:56:15.187: fh_io_ipc_msg: received FH_MSG_API_CLOSE client=27
*Jan 28 22:56:15.187: tty is now going through its death sequence

*Note "
```

debug event manager all

```
" is used to enable all debugs related to event manager.
```

Solución: de forma predeterminada, las políticas de EEM no se ejecutan durante más de 20 segundos. Si las acciones dentro del EEM tardan más de 20 segundos en ejecutarse, el EEM no puede completarse. Asegúrese de que el tiempo de ejecución del EEM es suficiente para permitir que se ejecuten las acciones del applet. Configure maxrun para especificar un valor de tiempo de ejecución máximo más apropiado.

Ejemplo:

<#root>

```
event manager
```

applet Capture

```
event timer watchdog time 120
```

maxrun 60

<-- Maxrun 60 specifies the capture will run for a maximum of 60 seconds.

```
action 1.0 cli command "enable"
action 1.1 cli command "no monitor capture CPUCapture"
action 2.0 cli command "monitor capture CPUCapture control-plane in match any buffer circular"
action 2.1 cli command "monitor capture CPUCapture start"
action 3.0 wait 45
```

<-- The altered maxrun allows the capture to run for the necessary time.

```
action 4.0 cli command "monitor capture CPUCapture stop"
action 4.1 cli command "show clock | append flash:CPUCapture.txt"
action 4.2 cli command "show mon cap CPUCapture buff dump | append flash:CPUCapture.txt"
action 5.0 syslog msg "CPUCapture Complete - Next capture in 2 minutes"
```

Problema: EEM se activa con demasiada frecuencia

A veces, varios casos de un determinado desencadenador se producen en un breve período de tiempo. Esto podría dar lugar a iteraciones excesivas del applet y tener consecuencias graves en el peor de los casos.

Este applet se desencadena en un patrón de syslog determinado, luego recopila la salida del comando show y agrega esta salida a un archivo. Específicamente, el applet se activa cuando el protocolo de línea cae para una interfaz identificada:

<#root>

```
event manager
```

applet MonitorLinkFlap

```
event syslog pattern "Interface GigabitEthernet1/0/23, changed state to down"
action 1.0 cli command "enable"
action 1.1 cli command "show ip route | append flash:MonitorLinkFlap.txt "
action 2.0 cli command "show interface gig1/0/23 | append flash:MonitorLinkFlap.txt"
action 3.0 cli command "show process cpu sorted | append flash:MonitorLinkFlap.txt"
action 4.0 cli command "show platform hardware fed active fwd-asic drops exceptions | append flash:Monit
action 5.0 syslog msg "Link has flapped - Data gathered"
```

El applet se activa cada vez que se observa el syslog. Un evento como una inestabilidad de interfaz puede ocurrir rápidamente en un corto período de tiempo.

<#root>

```
Switch#
```

sh log | in Data gathered

```
*Jan 29 04:19:06.678: %HA_EM-6-LOG: MonitorLinkFlap: Link has flapped - Data gathered
```

<-- The applet generates this syslog each time it fires.

```
*Jan 29 04:19:27.367: %HA_EM-6-LOG: MonitorLinkFlap: Link has flapped - Data gathered
*Jan 29 04:19:36.779: %HA_EM-6-LOG: MonitorLinkFlap: Link has flapped - Data gathered
*Jan 29 04:19:57.472: %HA_EM-6-LOG: MonitorLinkFlap: Link has flapped - Data gathered
```

```
*Jan 29 04:20:06.570: %HA_EM-6-LOG: MonitorLinkFlap: Link has flapped - Data gathered
*Jan 29 04:20:27.671: %HA_EM-6-LOG: MonitorLinkFlap: Link has flapped - Data gathered
*Jan 29 04:20:36.774: %HA_EM-6-LOG: MonitorLinkFlap: Link has flapped - Data gathered
*Jan 29 04:20:57.264: %HA_EM-6-LOG: MonitorLinkFlap: Link has flapped - Data gathered
```

El applet se ejecutó varias veces durante unos minutos, lo que dio lugar a un archivo de salida no deseado con datos extraños. El archivo también sigue aumentando de tamaño y continúa llenando los medios locales. Este simple ejemplo de EEM no plantea una gran amenaza operativa si se ejecuta repetidamente, pero este escenario potencialmente conduce a un desperfecto con scripts más complejos.

En esta situación, sería conveniente limitar la frecuencia con la que se activa el applet.

Solución: aplique un límite de velocidad para controlar la velocidad de ejecución de un applet. La palabra clave ratelimit se agrega a la instrucción trigger y se asocia a un valor en segundos.

Ejemplo:

<#root>

```
event manager
applet MonitorLinkFlap

event syslog pattern "Interface GigabitEthernet1/0/23, changed state to down"
ratelimit 60

<-- Ratelimit
```

specifies a minimum amount of time that must pass before the applet will again trigger.

```
action 1.0 cli command "enable"
action 1.1 cli command "show clock | append flash:MonitorLinkFlap.txt "
action 2.0 cli command "show interface gig1/0/23 | append flash:MonitorLinkFlap.txt"
action 3.0 cli command "show process cpu sorted | append flash:MonitorLinkFlap.txt"
action 4.0 cli command "show platform hardware fed active fwd-asic drops exceptions | append flash:Monit
action 5.0 syslog msg "Link has flapped - Data gathered"
```

Información Relacionada

Cisco IOS Embedded Event Manager 4.0

Prácticas recomendadas y guiones útiles para EEM

Acerca de esta traducción
Cisco ha traducido este documento combinando la traducción automática y los recursos humanos a fin de ofrecer a nuestros usuarios en todo el mundo contenido en su propio idioma.

Tenga en cuenta que incluso la mejor traducción automática podría no ser tan precisa como la proporcionada por un traductor profesional.

Cisco Systems, Inc. no asume ninguna responsabilidad por la precisión de estas traducciones y recomienda remitirse siempre al documento original escrito en inglés (insertar vínculo URL).