Fehlerbehebung und Test von EEM-Skripts

Inhalt

Einleitung

In diesem Dokument wird die Validierung von EEM-Skripts (Embedded Event Manager) beschrieben. Außerdem werden allgemeine betriebliche Überlegungen und Fehlerszenarien vorgestellt.

Voraussetzungen

Anforderungen

In diesem Dokument wird davon ausgegangen, dass der Leser bereits mit der Funktion des Cisco IOS/IOS XE Embedded Event Manager (EEM) vertraut ist. Wenn Sie mit dieser Funktion noch nicht vertraut sind, lesen Sie die EEM-Funktionsübersicht zuerst.

EEM auf den Catalyst Switches der 9000-Familie erfordert das DNA-Add-on für die Lizenzstufe "Network Essentials". Network Advantage unterstützt EEM vollständig.

Verwendete Komponenten

Die Informationen in diesem Dokument beziehen sich auf EEM Version 4.0, wie in der Catalyst-Switch-Familie implementiert.

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.

Hintergrundinformationen

EEM ist bei einer effektiven Bereitstellung eine nützliche Funktion, es muss jedoch sichergestellt werden,

dass das EEM genau das tut, was der Autor beabsichtigt. Unzureichend geprüfte Skripte können zu schwerwiegenden Problemen in der Produktion führen. Bestenfalls führt das Skript in unerwünschter Weise aus. In diesem Dokument finden Sie nützliche Informationen zum Testen und Überprüfen von EEM mit CLI-Befehlen zum Anzeigen von Befehlen. Außerdem werden einige gängige Fehlerszenarien und die zur Identifizierung und Behebung des Problems verwendeten Debugging-Methoden erläutert.

EEM-Validierung mit Show-Befehlen

Aktive Timer bestätigen

Wenn ein EEM-Skript bereitgestellt wird, das von einem Timer ausgelöst wird, stellen Sie sicher, dass der Timer aktiv ist und heruntergezählt wird, wenn das Skript nicht erwartungsgemäß ausgelöst wird.

Berücksichtigen Sie die EEM-Skripts test und test3:

<#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"
```

- Das erste Skript (Test) verwendet einen 60-sekündigen Watchdog-Zeitgeber, um das Skript zu starten.
- Das zweite Skript (test3) verwendet einen 300-Sekunden-Watchdog-Zeitgeber namens test3, um das Skript zu starten.

Konfigurierte Timer und der aktuelle Wert dieser Timer können mit dem Befehl **show event manager statistics server (Ereignismanager anzeigen) angezeigt werden**.

Beispiel

<#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'
```

Trigger-Ereignisse werden ausgelöst bestätigen

Wie im Abschnitt Confirm Timers are Active (Timer als aktiv bestätigen) dieses Dokuments erläutert, erhöht IOS XE die Spalte Triggered Events (Ausgelöste Ereignisse) für die Zeile des EEM-Applets-Clients in der Ausgabe des Show Event Manager Statistics Servers jedes Mal, wenn ein EEM-Applet ausgelöst wird. Um zu überprüfen, ob das EEM-Skript wie erwartet funktioniert, führen Sie das Triggerereignis mehrmals aus, und überprüfen Sie die Ausgabe des Statistikservers des Show Event Managers, um zu bestätigen, dass dieser Wert erhöht wird. Ist dies nicht der Fall, wurde das Skript nicht ausgelöst.

Wenn der Befehl mehrmals hintereinander ausgeführt wird, werden die Timerwerte heruntergezählt. Wenn der Timer den Wert 0 erreicht und das Skript ausgeführt wird, zählt auch die Anzahl der ausgelösten Ereignisse für EEM-Applets.

<#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
```

Hinweis: Sollte dies nicht der Fall sein, überprüfen Sie die konfigurierten Timer mithilfe Ihres Skripts.

Ereignisverlauf überprüfen

Bei Skripten, die nicht von Timern ausgelöst werden, ist der Befehl show event manager history events hilfreich, um zu bestätigen, dass Applets wie erwartet ausgelöst werden.

Betrachten Sie dieses EEM-Skript:

```
<#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

Dieses Skript wird ausgeführt, wenn der CLI-Ereignis-Manager "test_manual" ausgeführt wird, und gibt eine Syslog-Meldung aus. Neben der Ausgabe im Syslog kann die Ausführung dieses Skripts durch eine Überprüfung der Ausgabe von Ereignissen im Verlauf des Show Event Managers verifiziert werden:

<#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

EEM-Validierung mit manuellem Trigger

In einigen Szenarien ist es wünschenswert, ein EEM-Skript manuell auszulösen, um entweder den Ausführungsablauf zu testen oder eine einmalige Aktion auszuführen. Dies kann mit einem EEM-Skript mit einem Trigger für das Ereignis none erreicht werden, wie in der folgenden Ausgabe gezeigt:

<#root>

```
event manager
```

applet test_manual

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

Starten Sie das Skript manuell mit dem Befehl **event manager run test_manual** von der enable-Eingabeaufforderung aus:

```
<#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
```

Überlegungen zum Betrieb

Stellen Sie sicher, dass EEM-Skripts vor der Verwendung in der Produktion validiert werden. Im Allgemeinen gibt es einige primäre Möglichkeiten, wie ein Skript nicht wie erwartet funktioniert. Drei davon werden hier behandelt.

Dieser Abschnitt zeigt, wie Sie die folgenden drei häufigen Probleme mit EEM-Skripts erkennen:

1. CLI-Befehlsfehler: Der Befehl kann nicht analysiert und daher nicht ausgeführt werden.
2. Das Skript wird zu lange ausgeführt: EEM-Skripts sind standardmäßig auf eine Laufzeit von 20 Sekunden beschränkt. Wenn diese Zeit überschritten wird, wird das Skript beendet, bevor alle Befehle ausgeführt werden.
3. Das Skript wird zu oft ausgeführt: Manchmal kann das vom Skript verwendete Triggerereignis zu häufig auftreten, sodass das Skript schnell ausgelöst wird. Es ist wünschenswert zu steuern, wie oft und mit welcher Geschwindigkeit das Skript ausgelöst wird.

Problem: CLI-Befehle können nicht ausgeführt werden

Dieses Beispielskript enthält mehrere Probleme. Es handelt sich um ein einfaches Applet, das die Ausgabe mehrerer show-Befehle an eine Textdatei auf einem lokalen Flash-Medium anfügt:

```
<#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 | append
action 2.0 syslog msg  "Data Capture Complete"
```

Das Applet wurde erfolgreich ausgeführt, es wurden jedoch nicht die erwarteten Ergebnisse generiert:

<#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
```

Verwenden Sie die **Aktivitäts-CLI des eingebetteten Ereignismanagers debug**, um die Appletüberprüfung zu unterstützen.

<#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 ar

```
*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.

Fazit: Überprüfen Sie alle EEM-Aktionen ordnungsgemäß, und verwenden Sie Debugs, um Fehlkonfigurationen und typografische Fehler zu vermeiden.

Problem: EEM-Aktionen dauern länger als die maximale Laufzeit

In diesem Szenario wird ein einfaches EEM verwendet, um alle 120 Sekunden Paketerfassungen auf der Steuerungsebene zu sammeln. Es fügt neue Erfassungsdaten an eine Ausgabedatei an, die sich auf einem lokalen Speichermedium befindet.

<#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"
```

Sie können leicht feststellen, dass der EEM nicht wie erwartet abgeschlossen wird. Suchen Sie in lokalen Protokollen nach dem Syslog aus Aktion 5.0. Dieses Syslog wird bei jeder erfolgreichen Iteration des Applets gedruckt. Das Protokoll wurde nicht im Puffer gedruckt, und die Datei CPUCapture.txt wurde nicht in den Flash-Speicher geschrieben:

```
<#root>

Switch#

show logging | include "CPUCapture Complete"



Switch#

dir flash: | include CPUCapture.txt
```

Aktivieren Sie die zu untersuchenden Debugging-Funktionen. Das am häufigsten verwendete Debugging ist die **Aktionsaufforderung des Debugging-Ereignismanagers.** Dieses Dienstprogramm druckt einen Dialog der aufeinander folgenden Aktionen.

Debug-Ausgabe: Die Debug-Ausgabe zeigt das erfolgreich aufgerufene Applet an. Die ersten Aktionen laufen ohne Probleme, aber die Erfassung kann nicht abgeschlossen werden.

```
<#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.
```

Lösung: EEM-Richtlinien werden standardmäßig nicht länger als 20 Sekunden ausgeführt. Wenn die Ausführung der Aktionen innerhalb des EEM länger als 20 Sekunden dauert, wird der EEM nicht abgeschlossen. Stellen Sie sicher, dass die Laufzeit des EEM ausreicht, damit die Applet-Aktionen ausgeführt werden können. Konfigurieren Sie maxrun, um einen geeigneteren maximalen Laufzeitwert anzugeben.

Beispiel

<#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"
```

Problem: EEM löst zu häufig Probleme aus

Manchmal treten mehrere Instanzen eines bestimmten Auslösers in kurzer Zeit auf. Dies kann zu übermäßigen Iterationen des Applets führen und im schlimmsten Fall schwerwiegende Folgen haben.

Dieses Applet löst ein bestimmtes Syslog-Muster aus, erfasst die Ausgabe des Befehls show und hängt diese Ausgabe an eine Datei an. Insbesondere wird das Applet ausgelöst, wenn das Leitungsprotokoll für eine identifizierte Schnittstelle verloren geht:

<#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"
```

Das Applet wird jedes Mal ausgelöst, wenn das Syslog erkannt wird. Ein Ereignis wie ein Interface-Flapping kann schnell und in kurzer Zeit auftreten.

<#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
```

Das Applet lief im Laufe von wenigen Minuten mehrmals, was zu einer unerwünschten Ausgabedatei mit externen Daten führte. Die Datei wird außerdem immer größer und füllt weiterhin die lokalen Medien. Dieses einfache Beispiel für EEM stellt keine große betriebliche Bedrohung dar, wenn es wiederholt ausgeführt wird. Dieses Szenario führt jedoch möglicherweise zu einem Absturz mit komplexeren Skripts.

In diesem Szenario wäre es von Vorteil, die Auslösefrequenz des Applets zu begrenzen.

Lösung: Wenden Sie ein Ratenlimit an, um zu steuern, wie schnell ein Applet ausgeführt wird. Das ratelimit-Schlüsselwort wird an die Trigger-Anweisung angefügt und einem Wert in Sekunden zugeordnet.

Beispiel

<#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"
```

Zugehörige Informationen

Cisco IOS Embedded Event Manager 4.0

Best Practices und nützliche Skripte für EEM

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.