Understand Best Practices and Useful Scripts for EEM

Contents

Introduction

Prerequisites

- Requirements
- Components Used

Best Practices

- Confirm Appropriate Authentication is in Place
- Add Constraints for EEM Runtime and Rate-limit
- Avoid Out-of-order Execution
- Disable Pagination
- Design Scripts for Future Maintainability

Common EEM Logic Patterns

- Branch Code Paths with If/Else
- Loop Over Statements
- Extract Output via Regular Expressions (Regex)

Useful EEM Scripts

- Track Specific MAC Address for MAC Address Learn
- Monitor High CPU via SNMP OID
- Dynamically Match a PID and Record Stack Output
- Upgrade a Switch
- Dump Diagnostic Data to a File When an IP SLA Tracked Object Goes Down
- Send an Email from EEM
- Shutdown a Port on a Schedule
- Shutdown an Interface if a Given Packets Per Second (PPS) Rate is Reached
- Automatically Populate Interface Descriptions Using CDP Neighbors

Related Information

Introduction

This document describes Embedded Event Manager (EEM) script configuration best practices on Cisco IOS® XE devices.

Prerequisites

Requirements

Cisco recommends that you have knowledge of and familiarity with this topic:

- Cisco IOS and Cisco IOS XE Embedded Event Manager (EEM)

If you are not already familiar with this feature, please read the EEM Feature Overview first.

Components Used

The information in this document is based on these software and hardware versions:

- Cisco Catalyst 9300, 9400, and 9500 Switches
- Cisco IOS Software Version 16.X or 17.X



Note: These scripts are not supported by Cisco TAC and are provided on an as-is basis for educational purposes.

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. If your network is live, ensure that you understand the potential impact of any command.

Best Practices

This section covers some of the most common issues observed with the design and implementation of EEM scripts. For additional information on EEM best practices, see the EEM Best Practices document referenced under the References section.

Confirm Appropriate Authentication is in Place

If your device uses AAA, you must ensure that the EEM scripts configured on the device are either configured with an AAA user able to run the commands in the script, or that authorization bypass is configured with the command **authorization bypass** in the script definition.

Add Constraints for EEM Runtime and Rate-limit

By default, EEM scripts can run for a maximum of 20 seconds. If you design a script that takes longer to run, or has to wait between command execution, specify a **maxrun** value on the applet event trigger to alter

the default execution timer.

It is also important to consider how often the event that triggers your EEM script can run. If you trigger a script from a condition that occurs rapidly in a short amount of time (for example, syslog trigger for MAC flaps), it is important to include a rate-limit condition on your EEM script to prevent an excessive number of executions in parallel and prevent exhaustion of device resources.

Avoid Out-of-order Execution

As described in the EEM documentation, the order of execution for action statements is controlled by their label (for example, action 0001 cli command **enable** has a label of 0001). This label value is NOT a number, but rather alphanumeric. Actions are sorted in ascending alphanumeric key sequence, use the **label** argument as the sort key, and they are run in this sequence. This can lead to unexpected order of execution, based on how you structure your action labels.

Consider this example:

```
event manager applet test authorization bypass
event timer watchdog time 60 maxrun 60
action 13 syslog msg "You would expect to see this message first"
action 120 syslog msg "This message prints first"
```

Since 120 is before 13 in an alphanumeric comparison, this script does not run in the order you expect. To avoid this, it is useful to use a system of padding like this:

```
event manager applet test authorization bypass
event timer watchdog time 60 maxrun 60
action 0010 syslog msg "This message appears first"
action 0020 syslog msg "This message appears second"
action 0120 syslog msg "This message appears third"
```

Due to the padding here, the numbered statements evaluate in the expected order. The increment of 10 between each label allows for additional statements to be inserted into the EEM script later where needed, without the need to renumber all subsequent statements.

Disable Pagination

EEM looks for the device prompt to determine when command output is complete. Commands that output more data than can be displayed on one screen (as configured by your terminal length), can prevent EEM scripts from completion (and eventually killed via the maxrun timer) as the device prompt is not shown until all pages of the output are viewed. Configure **term len 0** at the start of EEM scripts that examine large outputs.

Design Scripts for Future Maintainability

When you design an EEM script, leave gaps between action labels to make it easier to update the EEM script logic in the future. When appropriate gaps are available (that is, two statements such as **action 0010** and **action 0020** leave a gap of 9 labels that can be inserted), new statements can be added as required without renumber or recheck of the action labels and ensure the actions continue to execute in the expected order.

There are common commands that you need to run at the beginning of your EEM scripts. This can include:

- set terminal length to 0
- enter enable mode
- enable automatic timestamp for command output

This is a common pattern in the examples shown in this document, where many of the scripts begin with the same 3 action statements to configure this.

Common EEM Logic Patterns

This section covers some common logic patterns and syntax blocks used in EEM scripts. The examples here are not complete scripts, but rather demonstrations of how specific functionality can be used to create complex EEM scripts.

Branch Code Paths with If/Else

EEM variables can be used to control the execution flow of EEM scripts. Consider this EEM script:

```
event manager applet snmp_cpu authorization bypass
event timer watchdog time 60
action 0010 info type snmp oid 1.3.6.1.4.1.9.9.109.1.1.1.1.3 get-type exact
action 0020 if $_info_snmp_value ge "50"
action 0030 syslog msg "This syslog message is sent if CPU utilization is above 50%"
action 0040 elseif $_info_snmp_value ge "30"
action 0050 syslog msg "This syslog message is sent if CPU utilization is above 30% and below 50%"
action 0060 else
action 0070 syslog msg "This syslog message is sent if CPU utilization is below 30%"
action 0080 end
```

This script runs every minute. Examine the value of the SNMP OID for CPU utilization, and then enter one of three different execution paths based on the value of the OID. Similar statements can be used on any other legal EEM variable to build complex execution flows in EEM scripts.

Loop Over Statements

Execution loops can be used to significantly shorten EEM scripts, and make them easier to reason about. Consider this script, designed to pull the interface statistics for Te2/1/15 6 times over a 1 minute period to check for small periods of high utilization:

```
event manager applet int_util_check auth bypass
event timer watchdog time 300 maxrun 120
action 0001 cli command "enable"
action 0002 cli command "term exec prompt timestamp"
action 0003 cli command "term length 0"
action 0010 syslog msg "Running iteration 1 of command"
action 0020 cli command "show interface te2/1/15 | append flash:interface_util.txt"
action 0030 wait 10
action 0040 syslog msg "Running iteration 2 of command"
action 0050 cli command "show interface te2/1/15 | append flash:interface_util.txt"
action 0060 wait 10
action 0070 syslog msg "Running iteration 3 of command"
action 0080 cli command "show interface te2/1/15 | append flash:interface_util.txt"
action 0090 wait 10
action 0100 syslog msg "Running iteration 4 of command"
action 0110 cli command "show interface te2/1/15 | append flash:interface_util.txt"
action 0120 wait 10
action 0130 syslog msg "Running iteration 5 of command"
action 0140 cli command "show interface te2/1/15 | append flash:interface_util.txt"
action 0150 wait 10
action 0160 syslog msg "Running iteration 6 of command"
action 0170 cli command "show interface te2/1/15 | append flash:interface_util.txt"
```

With EEM loop constructs, this script can be significantly shortened:

```
event manager applet int_util_check auth bypass
event timer watchdog time 300 maxrun 120
action 0001 cli command "enable"
action 0002 cli command "term exec prompt timestamp"
action 0003 cli command "term length 0"
action 0010 set loop_iteration 1
action 0020 while $loop_iteration le 6
action 0030 syslog msg "Running iteration $loop_iteration of command"
action 0040 cli command "show interface te2/1/15 | append flash:interface_util.txt"
action 0050 wait 10
action 0060 increment loop_iteration 1
action 0070 end
```

Extract Output via Regular Expressions (Regex)

The EEM regexp statement can be used to extract values from command output to be used in subsequent commands and enable dynamic command creation within the EEM script itself. Refer to this code block for an example to extract the SNMP ENGINE PID from the output of **show proc cpu | i SNMP engine** and print it to a syslog message. This extracted value can also be used in other commands that require a PID to run.

```
event manager applet check_pid auth bypass
event none
action 0010 cli command "show proc cpu | i SNMP ENGINE"
action 0020 regexp "^[ ]*([0-9]+) .*" $_cli_result match match1
action 0030 syslog msg "Found SNMP Engine PID $match1"
```

Useful EEM Scripts

Track Specific MAC Address for MAC Address Learn

In this example, the MAC address b4e9.b0d3.6a41 is tracked. The script checks every 30 seconds to see if the specified MAC address has been learned in the ARP or MAC tables. If the MAC is seen, the script takes these actions:

- outputs a syslog message (This is useful when you want to confirm where a MAC address is learned, or when/how often it is learned).

Implementation

```
event manager applet mac_trace authorization bypass
event timer watchdog time 30
action 0001 cli command "enable"
action 0002 cli command "term exec prompt timestamp"
action 0003 cli command "term length 0"
action 0010 cli command "show ip arp | in b4e9.b0d3.6a41"
action 0020 regexp ".*(ARPA).*" $_cli_result
action 0030 if $_regexp_result eq 1
action 0040 syslog msg $_cli_result
action 0050 end
action 0060 cli command "show mac add vlan 1 | in b4e9.b0d3.6a41"
action 0070 regexp ".*(DYNAMIC).*" $_cli_result
action 0080 if $_regexp_result eq 1
action 0090 syslog msg $_cli_result
action 0100 end
```

Monitor High CPU via SNMP OID

This script monitors a SNMP OID used to read the CPU busy percentage in the last 5 seconds. When the CPU is over 80% busy, the script takes these actions:

- Creates a timestamp from the output of show clock, and uses this to create a unique filename.
- Outputs about process and software state are then written to this file.
- An Embedded Packet Capture (EPC) is configured to capture 10 seconds of traffic destined to the

control-plane and writes it to a file.
- Once EPC capture is completed, the EPC configuration is removed, and the script exits.

Implementation

```
event manager applet high-cpu authorization bypass
event snmp oid 1.3.6.1.4.1.9.9.109.1.1.1.1.3 get-type next entry-op gt entry-val 80 poll-interval 1 rat
action 0001 cli command "enable"
action 0002 cli command "term exec prompt timestamp"
action 0003 cli command "term length 0"
action 0010 syslog msg "High CPU detected, gathering system information."
action 0020 cli command "show clock"
action 0030 regex "([0-9]|[0-9][0-9]):([0-9]|[0-9][0-9]):([0-9]|[0-9][0-9])" $_cli_result match match1
action 0040 string replace "$match" 2 2 "."
action 0050 string replace "$_string_result" 5 5 "."
action 0060 set time $_string_result
action 0070 cli command "show proc cpu sort | append flash:tac-cpu-$time.txt"
action 0080 cli command "show proc cpu hist | append flash:tac-cpu-$time.txt"
action 0090 cli command "show proc cpu platform sorted | append flash:tac-cpu-$time.txt"
action 0100 cli command "show interface | append flash:tac-cpu-$time.txt"
action 0110 cli command "show interface stats | append flash:tac-cpu-$time.txt"
action 0120 cli command "show log | append flash:tac-cpu-$time.txt"
action 0130 cli command "show ip traffic | append flash:tac-cpu-$time.txt"
action 0140 cli command "show users | append flash:tac-cpu-$time.txt"
action 0150 cli command "show platform software fed switch active punt cause summary | append flash:tac
action 0160 cli command "show platform software fed switch active cpu-interface | append flash:tac-cpu-
action 0170 cli command "show platform software fed switch active punt cpuq all | append flash:tac-cpu-
action 0180 cli command "no monitor capture tac_cpu"
action 0190 cli command "monitor capture tac_cpu control-plane in match any file location flash:tac-cpu
action 0200 cli command "monitor capture tac_cpu start" pattern "yes"
action 0210 cli command "yes"
action 0220 wait 10
action 0230 cli command "monitor capture tac_cpu stop"
action 0240 cli command "no monitor capture tac_cpu"
```

Dynamically Match a PID and Record Stack Output

This script looks for a syslog message that the SNMP input queue is full and takes these actions:

- Records the output of **show proc cpu sort** to a file.
- Extracts the PID of the SNMP ENGINE process via regex.
- Uses the SNMP PID in subsequent commands to obtain the stack data for the PID.
- Removes the script from the configuration so no more executions of it occur.

Implementation

```
event manager applet TAC-SNMP-INPUT-QUEUE-FULL authorization bypass
event syslog pattern "INPUT_QFULL_ERR" ratelimit 40 maxrun 120
action 0010 cli command "en"
```

```
action 0020 cli command "show proc cpu sort | append flash:TAC-SNMP.txt"
action 0030 cli command "show proc cpu | i SNMP ENGINE"
action 0040 regexp "^[ ]*([0-9]+) .*" $_cli_result match match1
action 0050 syslog msg "Found SNMP Engine PID $match1"
action 0060 cli command "show stacks $match1 | append flash:TAC-SNMP.txt"
action 0070 syslog msg "$_cli_result"
action 0080 cli command "configure terminal"
action 0090 cli command "no event manager applet TAC-SNMP-INPUT-QUEUE-FULL"
action 0100 cli command "end"
```

Upgrade a Switch

This script is configured to pattern match on the non-standard prompt returned by the **install add file <file> activate commit** command and respond to the prompts. No trigger event is configured, so the EEM script must be manually triggered by a user when the upgrade needs to occur via **event manager run UPGRADE**. The maxrun timer is set for 300 seconds instead of the default value of 20 seconds as the **install add** command takes a significant amount of time to run.

Implementation

```
event manager applet UPGRADE authorization bypass
event none maxrun 300
action 0001 cli command "enable"
action 0002 cli command "term length 0"
action 0020 cli command "install add file flash:cat9k_iosxe.16.06.02.SPA.bin activate commit" pattern "
action 0030 cli command "y" pattern "y\/n"
action 0040 syslog msg  "Reloading device to upgrade code"
action 0050 cli command "y"
```

Dump Diagnostic Data to a File When an IP SLA Tracked Object Goes Down

This script is triggered when IP SLA object 11 goes down and takes these actions:

- Collect MAC table, ARP table, syslogs, and Routing table.
- Write info to a file on flash: called sla_track.txt.

Implementation

```
ip sla 10
icmp-echo 10.10.10.10 source-ip 10.10.10.10
frequency 10
exit
ip sla schedule 10 life forever start-time now
track 11 ip sla 10 reachability
exit
```

```
event manager applet track-10 authorization bypass
event track 11 state down
action 0001 cli command "enable"
action 0002 cli command "term exec prompt timestamp"
action 0003 cli command "term length 0"
action 0010 syslog msg  "IP SLA object 10 has gone down"
action 0020 cli command "show mac address-table detail | append flash:sla_track.txt"
action 0030 cli command "show ip arp | append flash:sla_track.txt"
action 0040 cli command "show log | append flash:sla_track.txt"
action 0050 cli command "show ip route | append flash:sla_track.txt"
```

Send an Email from EEM

This script is triggered when the pattern described in the event syslog pattern statement is seen, and takes these actions:

- Sends an email from an internal email server (this assumes that the internal email server allows for open authentication from the device).

Implementation

```
event manager environment email_from email_address@company.test
event manager environment email_server 192.168.1.1
event manager environment email_to dest_address@company.test
event manager applet email_syslog
event syslog pattern "SYSLOG PATTERN HERE” maxrun 60
action 0010 info type routername
action 0020 mail server "$email_server" to "$email_to" from "$email_from" subject "SUBJECT OF EMAIL - Sy
```

Shutdown a Port on a Schedule

This script shuts down port Te2/1/15 every day at 6PM.

Implementation

```
event manager applet shut_port authorization bypass
event timer cron cron-entry "0 18 * * *"
action 0001 cli command "enable"
action 0002 cli command "term exec prompt timestamp"
action 0003 cli command "term length 0"
action 0010 syslog msg  "shutting port Te2/1/15 down"
action 0030 cli command "config t"
action 0040 cli command "int Te2/1/15"
action 0050 cli command "shutdown"
action 0060 cli command "end"
```

Shutdown an Interface if a Given Packets Per Second (PPS) Rate is Reached

This script checks the PPS rate on interface Te2/1/9 in the TX direction every second. If the PPS rate exceeds 100, it takes these actions:

- Logs the show int output for the interface to syslog.
- Shuts down the interface.

Implementation

```
event manager applet disable_link authorization bypass
event interface name te2/1/9  parameter  transmit_rate_pps entry-op ge entry-val 100 poll-interval 1 en
action 0001 cli command "enable"
action 0002 cli command "term length 0"
action 0010 syslog msg  "Detecting high input rate on interface te2/1/9.  Shutting interface down."
action 0020 cli command "show int te2/1/9"
action 0030 syslog msg  $_cli_result
action 0040 cli command "config t"
action 0050 cli command "int te2/1/9"
action 0060 cli command "shutdown"
action 0070 cli command "end"
```

Automatically Populate Interface Descriptions Using CDP Neighbors

This script periodically checks CDP neighbor information and automatically updates local interface descriptions based on the connected CDP neighbor device. The script runs every 5 minutes and takes these actions:

- Executes show cdp neighbors detail.
- Parses the Device ID and local interface.
- Enters interface configuration mode for each discovered interface.
- Sets the interface description to Connected to <CDP Neighbor>.
- Generates a syslog message confirming the update..

This is useful for maintaining accurate interface descriptions in environments where links change frequently or where consistent documentation is required.

Implementation

```
event manager applet CDP-Neighbor-Description
```

```
event timer watchdog time 300
event none
action 1.0 cli command "enable"
action 2.0 cli command "show cdp neighbors detail"
action 3.0 set temp_device ""
action 4.0 foreach line "$_cli_result" "\n"
action 4.1 regexp "Device ID: ([^\r\n]+)" "$line" match device_id
action 4.2 if $_regexp_result eq "1"
action 4.3 set temp_device "$device_id"
action 4.4 end
action 4.5 regexp "Interface: ([^,]+)," "$line" match intf
action 4.6 if $_regexp_result eq "1"
action 4.7 if "$temp_device" ne ""
action 4.71 cli command "configure terminal"
action 4.72 cli command "interface $intf"
action 4.73 cli command "description Connected to $temp_device"
action 4.74 cli command "end"
action 4.75 syslog msg "EEM: Set description on $intf to 'Connected to $temp_device'"
action 4.76 set temp_device ""
action 4.8 end
action 4.9 end
action 5.0 end
```

Related Information

- Cisco EEM Best Practices
- Cisco Technical Support & Downloads