CHAPTER 1

Configuring the Common Criteria Tcl Scripts

To monitor the packet drop event on the ASR 1000 Series Router, use the Common Criteria Tcl scripts.

This chapter includes the following sections:

Common Criteria Tcl Scripts Overview

Common Criteria (CC) is an international standard for evaluating IT product security and reliability. It is recognized by over 15 countries around the world including Australia, Canada, France, Germany, Greece, Italy, Japan, New Zealand, Spain, UK, South Korea and the United States. Many government customers around the world consider Common Criteria a mandatory requirement for purchasing network security products.

Common Criteria is a methodology for product evaluation. There are seven levels of evaluation and only levels 1 through 4 are mutually recognized by the participating countries. Products typically target EAL2 or EAL4, an evaluation conducted in any one of the participating countries is valid for the rest for the members. Cisco continues to be a global leader in completing and pursuing Common Criteria evaluations.

ASR1000 Series Routers support packet drop event monitoring as required by the Common Criteria standards. The Common Criteria features can be enabled using Tcl scripting. To find out more about Cisco IOS XE scripting using Tcl, see the “Cisco IOS XE Scripting with Tcl” chapter of the Cisco IOS XE Network Management Software Configuration Guide.

Common Criteria leverages the IOS XE Embedded Syslog Manager (ESM) and Embedded Event Manager (EEM) mechanisms for enabling periodic actions. The ESM feature provides a programmable framework that allows you to filter, escalate, correlate, route, and customize system logging messages prior to delivery by the Cisco IOS system message logger. For more information, see the Embedded Syslog Manager (ESM) Configuration Guide.

Table 1-1 lists the Common Criteria Tcl scripts.

Table 1-1 Common Criteria Tcl Scripts

Script Name	Description
timer.tcl	Supports the Timer events for other scripts.
alarms_db.tcl	Manages the alarms database.
em_ike_phase1_failure.tcl	Monitors the Internet Key Exchange (IKE) protocol Phase 1 negotiation failures.
em_ike_phase2_failure.tcl	IKEv1 Phase 2 negotiation failures watcher script.
em_login_failure.tcl	Monitors the user login failures.
em_monitor_violation.tcl	Monitors the information flow violations. ACL-based event monitors must be configured to trigger the violation monitor watcher.
em_monitor_vpn_event.tcl	Monitors the VPN encryption, decryption faults, and packet replay events.
monitor_ipsec.tcl	Configures the VPN event monitors.
syslog_exclude.tcl	Excludes the syslog messages containing the keywords from the syslog database.
syslog_include.tcl	Includes the syslog messages containing the keywords in the syslog database.
esm_conf_vty.tcl	Configures the syslog message output to the connected vty devices.

Installing the Common Criteria Tcl Scripts

Super administrator can copy the scripts from a portable device such as a USB flash drive on the hard disk, which is defined as a protected directory.

Example:

```
Copy bootflash:<folder name> <Tcl file name> harddisk:/cc_scripts
```

How to Configure the Common Criteria Tcl Scripts

To configure the Common Criteria Tcl scripts, complete the following steps:

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **logging filter** [*script-url*] [**args** *filter-arguments*]
4. **end**
5. **show logging**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables the privileged EXEC mode. Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters the global configuration mode.
Step 3	**logging filter** [*script-url*] [**args** *filter-arguments*] **Example:** `Router(config)# logging filter bootflash:/escalate.tcl 1 args CONFIG_I 1`	Specifies one or more syslog filter modules to be applied to the generated system logging messages. To remove a module from the list of modules to be executed, use the **no** form of this command. **Note** Repeat this command for each syslog filter module that should be used. • (Optional) The *script-url* argument specifies the script URL. **Note** Provide a valid directory location, an incorrect location can trigger a router reload. • (Optional) The **args** *filter-arguments* syntax can be added to pass arguments to the specified filter. Multiple arguments can be specified. The number and type of arguments should be defined in the syslog filter module. For example, if the syslog filter module is designed to accept a specific e-mail address as an argument, you could pass the e-mail address using the **args user@host.com** syntax. Multiple arguments are typically delimited by spaces.
Step 4	**end** **Example:** `Router(config)# end`	Ends your current configuration session and returns the CLI to privileged EXEC mode.
Step 5	**show logging** **Example:** `Router# show logging`	(Optional) Displays the status of system logging, including the status of ESM-filtered logging: • If filtered logging to the buffer is enabled, this command also displays the data stored in the buffer. • The order in which the syslog filter modules are listed in the output of this command is the order in which the filter modules are executed.

Examples

This section provides the following configuration examples:

Alarm Confirmation Timer

This Common Criteria alarm confirmation timer watches for repetitive CC alarm confirmation requests. These requests are managed by the **timer.tcl** script:

```
logging filter <script-url>timer.tcl [args <interval>]
```

`interval`—interval between two successive CC alarm prompts. A default interval between two successive CC alarm prompts is 60 seconds:

```
logging filter bootflash:timer.tcl args 120
```

Alarm Database Manager

This Common Criteria alarm database manager maintains a repository of unconfirmed CC alarms. This request is managed by the **alarms_db.tcl** script:

```
logging filter <script-url>alarms_db.tcl [args <audible-property>]
```

`audible-property`—`alarm_audible` or `alarm_not_audible`

When the `alarm-property` is set to `alarm_audible`, it enables audio signals for every CC alarm confirmation prompt. By default, `audible-property` is set to `alarm_not_audible`:

```
logging filter bootflash:alarms_db.tcl args alarm_audible
```

IKEv1 Phase 1 and Phase 2 Failures Catcher

The IKEv1 failure catcher alert enables the monitoring of IKEv1 phase 1 and phase 2 negotiation failures. The commands for configuring the IKEv1 negotiation failure monitor are:

```
logging filter <script-url>em_ike_phase1_failure.tcl [args threshold [interval]]
logging filter <script-url>em_ike_phase2_failure.tcl [args threshold [interval]]
```

The argument values are as follows:

- `Threshold`—number of failures after which the CC alarm is raised. The default threshold value is 1.

- `Interval`—time interval during which the number of failures must reach a set threshold. On reaching the threshold, the alarms are triggered. The default value is indefinite.

  If the interval value is not set, the CC alarm is raised after the threshold pertaining to the number of failures is crossed.

  If the interval value is set, and the value is less than the threshold value, the failure counter is reset and the CC alarm is not raised.

Example:

```
logging filter bootflash:em_ike_phase1_failure.tcl args  3 300
```

This configuration raises a CC alarm after three IKEv1 Phase 1 failures occur during the 300-second interval.

If the number of failures are less than three within the 300-second interval, the CC alarm is not raised, and the failure counter is reset.

Syslog Filter

Syslog filter commands support both inclusive and exclusive filtering of syslog messages. The configured filters determine the order of syslog command execution. The number of syslog filters that can be configured depends on the device memory size.

The commands for configuring the syslog filters are:

- Inclusive filtering:

  ```
  logging filter <script-url>syslog_include.tcl [args <string>]
  ```

  The value of the `string` argument is an arbitrary character string.

  Example:

  ```
  logging filter bootflash:syslog_include.tcl args  ALARM
  logging filter bootflash:syslog_include.tcl args  LINK
  ```

  Syslog messages containing character strings such as ALARM or LINK are propagated to the configured auditable events repositories. Syslog messages that do not contain the configured character strings are dropped.

- Exclusive filtering

  ```
  logging filter <script-url>syslog_exclude.tcl [args <string>]
  ```

  The value of the string argument is an arbitrary character string.

  Example:

  ```
  logging filter bootflash:syslog_exclude.tcl args ALARM
  ```

  Any syslog message that contains the configured character string is dropped. Syslog messages that do not contain the configured character string are propagated to the auditable events repository.

Note Strings containing special characters should be enclosed within a pair of the escape characters such as single quotes (' '), double quotes (" "), or backslash (\ \).

Information Flow Violations Watcher

When an information flow violation occurs, the information flow violations watcher triggers a CC alarm. The command to configure the information flow violations watcher is:

```
logging filter <script-url>em_monitor_violation.tcl
```

IPsec Policy Violation Category Watcher

When an IPsec policy violation occurs, the IPsec policy violations watcher triggers a CC alarm. The command to configure IPsec policy violations watcher is

```
logging filter <script-url>monitor_ipsec.tcl args <esp> <category> <threshold>
```

The argument values are as follows:

- `esp`—Active or standby ASR1000 ESP on which IPsec policy violations are monitored.
- `category`—`decrypt-failed`, or `encrypt-failed`, or `replay`.

  Watches for decryption or encryption failures or IPsec packets replay events
- `threshold`—Number of events watched after which a cumulative event is reported. The threshold value must be greater than 0.

Note All command arguments are mandatory.

Multiple command lines can be configured for watching multiple categories of the IPsec policy violations.

Example:

```
logging filter bootflash:monitor_ipsec.tcl args active replay 100000
```

This command line configures a watcher for the IPsec packet replay violations. The watcher triggers an alarm after 100000 replayed IPsec packets are detected.

VPN Policy Violations Catcher

The VPN policy violations catcher triggers an alert if a violation occurs on the previously configured VPN policy:

```
logging filter <script-url> em_monotor_vpn_event.tcl
```

Replication Output of Syslog Messages

To replicate the syslog messages to all the connected terminal devices, use the following command:

```
logging filter <script-url>esm_conf_vty.tcl
```

Generating the Event Alarm Reports

CC Protection Profiles identify a number of events that generate alarms. The alarms must be acknowledged by the administrator.

For example, the following commands display the acknowledgement of alarms on the router:

```
000077:  *Apr 21 03:02:19.566: %CC-6-INFO: Please confirm alarm 000077
000077:  *Apr 21 02:54:23.001: %CC-6-ALARM: Login Authentication Failed for user eve 2
times in 11 seconds interval
```

Alarm confirmation on the router:

```
Router (config)#event manager environment confirm_alarm 000077
```

Based on the administrator-specified values, the syslog messages indicate alarm-inducing events.The reports that are generated for the event alarms include:

- Specified number of authentication failures—IOS supports logging of authentication events. To report authentication failures, administrators use the following commands:
  - `conf t`
  - `login on-failure log`
  - `end`
- Specified number of information flow policy violations by:
  - Individual source network identifiers, such as IP address, within a specified time.
  - Individual destination network identifiers, within user-specified time.
  - Individual destination subject service identifiers, such as TCP port, within user-specified time.
  - Individual or group rules within user-specified time.

Note The **monitor drop** command is used to configure event monitoring of the information flow policy violations.

- The VPN policy violation catcher includes:
  - Any detected replay of TSF data or security attributes
  - Security administrator-specified number of encryption failures
  - Security administrator-specified number of decryption failures

Note The **set platform hardware qfp feature ipsec event-monitor** command is used to configure VPN-specific event monitoring.
The **clear platform hardware qfp <mastership> feature ipsec event-monitor** command is used for removing the event monitors.

Configuration Examples of the Common Criteria Tcl Scripts

This section provides the following Tcl script examples:

- Example: Tcl Scripts for Common Criteria Alarms, page 1-8
- Example: Tcl Scripts for the IKEv1 Phase 1 Failure Catcher, page 1-12

Example: Tcl Scripts for Common Criteria Alarms

```
namespace eval ::common_criteria_alarms {
    # namespace variables
    array set unconfirmed_alarms_db {}
    array set logged_in_users_info {}
    array set alarms_linked_list {}
    variable first_alarm_id
    variable last_alarm_id

    array set msgs_to_watch {
        CC-6-ALARM    1
        CC-6-TIMER    1
        PARSER-5-CFGLOG_LOGGEDCMD  1
        SEC_LOGIN-5-LOGIN_SUCCESS 1
    }

    array set login_success_msg {
        SEC_LOGIN-5-LOGIN_SUCCESS 1
    }

    array set msg_to_log {
        CC-6-ALARM    1
    }

    array set msg_to_confirm {
        PARSER-5-CFGLOG_LOGGEDCMD  1
    }

    array set msg_timer {
        CC-6-TIMER    1
    }

    # Should I process this message ?
    proc query_category {cat} {
        variable msgs_to_watch

        if { [info exists msgs_to_watch($cat)] } {
            return $msgs_to_watch($cat)
        } else {
            return 0
        }
    }
```

```
# Should I log this message ?
proc query_log {cat} {
    variable msg_to_log

    if { [info exists msg_to_log($cat)] } {
        return $msg_to_log($cat)
    } else {
        return 0
    }
}

# Should I log this message ?
proc query_is_login_success {cat} {
    variable login_success_msg

    if { [info exists login_success_msg($cat)] } {
        return $login_success_msg($cat)
    } else {
        return 0
    }
}

# Should I confirm this message ?
proc query_confirm {cat} {
    variable msg_to_confirm

    if { [info exists msg_to_confirm($cat)] } {
        return $msg_to_confirm($cat)
    } else {
        return 0
    }
}

# is this timer syslog?
proc query_timer {cat} {
    variable msg_timer

    if { [info exists msg_timer($cat)] } {
        return $msg_timer($cat)
    } else {
        return 0
    }
}

# Accept alarm string and generate a syslog
proc generate_syslog {alarm_msg} {

# store all current syslog global params
    set prev_orig_msg       $::orig_msg
    set prev_timestamp      $::timestamp
    set prev_facility       $::facility
    set prev_mnemonic       $::mnemonic
    set prev_severity       $::severity
    set prev_stream         $::stream
    set prev_traceback      $::traceback
    set prev_pid            $::pid
    set prev_process        $::process
    set prev_format_string  $::format_string
    set prev_msg_args       $::msg_args

    # construct a new syslog with the details of the login failure
    # alarm

    set ::timestamp  [cisco_service_timestamp]
```

```
        set ::facility "CC"
        set ::mnemonic "INFO"
        set ::severity 6
        set ::stream 2
        set ::traceback "cc_internal_syslog"
        set ::pid ""
        set ::process ""
        set ::format_string ""
        set ::msg_args {}
        set ::orig_msg [format "%s %s: %s%s-%d-%s: %s" $::buginfseq $::timestamp "%"
$::facility $::severity $::mnemonic $alarm_msg]

        # Send a syslog to be catched by the script that handles the
        # alarms
        esm_errmsg 0

        # restore all syslog global params

        set ::orig_msg        $prev_orig_msg
        set ::timestamp       $prev_timestamp
        set ::facility        $prev_facility
        set ::mnemonic        $prev_mnemonic
        set ::severity        $prev_severity
        set ::stream          $prev_stream
        set ::traceback       $prev_traceback
        set ::pid             $prev_pid
        set ::process         $prev_process
        set ::format_string   $prev_format_string
        set ::msg_args        $prev_msg_args
    }


    # Process all alarm related syslogs (new alarm/timer/confirm)

    proc process_syslog {} {
        variable unconfirmed_alarms_db
        variable alarms_linked_list
        variable logged_in_users_info
        variable first_alarm_id
        variable last_alarm_id
        variable alarm_to_confirm

        # empty msg?
        if { [string length $::orig_msg] == 0} {
            return ""
        }

        set category "$::facility-$::severity-$::mnemonic"

        # Should I process this syslog?
        set need_to_process [query_category $category]
        if { $need_to_process == 0 } {
            return $::orig_msg
        }

        # Is this a login success syslog?
        set is_login_success [query_is_login_success $category]
        if { $is_login_success == 1} {
            # Save the ip address aln local port of the logged in user
            set user [lindex $::msg_args 0]
            set ip_address [lindex $::msg_args 1]
            set local_port [lindex $::msg_args 2]
            if { $ip_address == "0.0.0.0"} {
                set ip_address "console"
```

```
                set local_port "console"
            }
            set logged_in_users_info($user) [list $ip_address $local_port]
            return $::orig_msg
        }

        # Should I log this msg in unconfirmed alarms database?
        set need_to_log [query_log $category]
        if { $need_to_log == 1 } {
            set alarm_id [lindex [split $::buginfseq :]  0]
            if { $alarm_id == "" } {
                set alarm_db_syslog "Alarm Id is missing. Please configure 'service
sequence-numbers'"
                generate_syslog $alarm_db_syslog
                return ""
            }
            # Update the alarms linked list
            if { [info exists last_alarm_id] } {
                set alarms_linked_list($last_alarm_id) $alarm_id
                set last_alarm_id $alarm_id
                set alarms_linked_list($last_alarm_id) "void"
            } else {
                set first_alarm_id $alarm_id
                set last_alarm_id $alarm_id
                set alarms_linked_list($last_alarm_id) "void"
            }

            set unconfirmed_alarms_db($alarm_id) "$::orig_msg"
            return $::orig_msg
        }

        # Is this a confirmation syslog?
        set need_to_confirm [query_confirm $category]
        if { $need_to_confirm == 1 } {
            if { "event manager environment confirm_alarm" == [lrange [split [lindex
$::msg_args 1]] 0 3]} {
                set alarm_id_to_confirm [lindex [split [lindex $::msg_args end]] end]
                if { [info exists alarms_linked_list($alarm_id_to_confirm)] == 0} {
                    set alarm_db_syslog "ERROR: alarm id $alarm_id_to_confirm does not
exist"
                    generate_syslog $alarm_db_syslog
                    return $::orig_msg
                }

                if { $alarm_id_to_confirm != $first_alarm_id } {
                   set alarm_db_syslog "ERROR: Only the displayed alarm ($first_alarm_id)
can be confirmed"
                    generate_syslog $alarm_db_syslog
                    return $::orig_msg
                }

                set alarm_to_confirm unconfirmed_alarms_db($alarm_id_to_confirm)

                if { [string length $alarm_to_confirm] != 0} {
                    unset unconfirmed_alarms_db($alarm_id_to_confirm)

                    # Get the next alarm id to confirm
                    set first_alarm_id $alarms_linked_list($first_alarm_id)
                    unset alarms_linked_list($alarm_id_to_confirm)

                    if { $first_alarm_id == "void" } {
                        unset first_alarm_id
                        unset last_alarm_id
                    }
```

```
                    # Add user location info (ip adderss and local port
                    # to the orig msg
                    set user [lindex $::msg_args 0]
                    if { [info exists logged_in_users_info($user)] == 0 } {
                        set location_info {"unknown" "unknown"}
                    } else {
                        set location_info  $logged_in_users_info($user)
                    }
                    set ip_address [lindex $location_info 0]
                    set local_port [lindex $location_info 1]
                    set new_orig_msg "$::orig_msg \[source: $ip_address\] \[local_port:
$local_port\]"
                    set ::orig_msg $new_orig_msg
                    # Are there any unconfirmed alarms?
                    if { [info exists first_alarm_id] } {
                        set first_alarm_msg $unconfirmed_alarms_db($first_alarm_id)
                        set audible_sound ""
                        if { [lindex $::cli_args 0] == "alarm_audible" } {
                            set audible_sound "\a\t\a\t\a\t\a"
                        }
                        set alarm_db_syslog "Please confirm alarm $first_alarm_id \n
$first_alarm_msg $audible_sound"
                        generate_syslog $alarm_db_syslog
                    }
                    return $::orig_msg
                }
            } else {
                return $::orig_msg
            }
        }

        # Is this a cron/timer msgs
        set is_timer_msg [query_timer $category]
        if { $is_timer_msg == 1 } {
           # Are there any unconfirmed alarms?
            if { [info exists first_alarm_id] } {
                set first_alarm_msg $unconfirmed_alarms_db($first_alarm_id)
                set audible_sound ""
                if { [lindex $::cli_args 0] == "alarm_audible" } {
                    set audible_sound "\a\t\a\t\a\t\a"
                }
                set alarm_db_syslog "Please confirm alarm $first_alarm_id \n
$first_alarm_msg $audible_sound"
                generate_syslog $alarm_db_syslog
            }
        }

        return ""
    }

    # Process the message
    process_syslog

} ;# end namespace common_criteria_alarms
```

Example: Tcl Scripts for the IKEv1 Phase 1 Failure Catcher

```
namespace eval ::ike_auth_failures {
    # namespace variables
```

```
array set host_failed_ike_auth {}

# Should I process this message ?
proc query_category {cat} {
    variable msg_to_watch

    if { [info exists msg_to_watch($cat)] } {
        return 1
    } else {
        return 0
    }
}

# handle ike authentication failure for a given ip address
proc process_ike_auth_failure {ipaddress} {
    variable  host_failed_ike_auth

    set current_time    [clock seconds]

    #default values, change if get as cli args
    set alarm_threshold  1
    set time_gap         0

    # Get the list timestamps of previous ike authentication failures
    # of this ipaddress
    set time_list $host_failed_ike_auth($ipaddress)
    set list_len [llength $time_list]

    # First arg (if exists) is alarm threshlod
    if { [info exists ::cli_args] == 1 } {
        set alarm_threshold [lindex $::cli_args 0]

        # Second arg (if exists) is time gap
        if { [llength $::cli_args] > 1 } {
            set time_gap [lindex $::cli_args 1]
        }
    }

    if { $time_gap != 0 } {
        set i 0
        # run though the list and keep only the timestamps which are still
        # in the given time gap from current time
        while  { $i < $list_len } {
            if {[expr $current_time - [lindex $time_list $i]] <= $time_gap } {
                lappend new_time_list [lindex $time_list $i]
            }
            incr i
        }

        # update the timestamp list for the given ipaddress
        set host_failed_ike_auth($ipaddress) $new_time_list
    } else {
        set new_time_list $host_failed_ike_auth($ipaddress)
    }

    # does the updated timestamp list has more items than the threshold
    if { [llength $new_time_list] >= $alarm_threshold } {
        if { $time_gap != 0 } {
            # Need to send a new alarm.
            set ike_auth_fail_msg [format "Ike authentication failed with %s %d times
in %d seconds interval" $ipaddress [llength $new_time_list] [expr $current_time - [lindex
$new_time_list 0]]]
        } else {
```

```
                set ike_auth_fail_msg [format "Ike authentication failed with %s %d times"
$ipaddress [llength $new_time_list]]
            }
            # store all current syslog global params
            set prev_orig_msg       $::orig_msg
            set prev_timestamp      $::timestamp
            set prev_facility       $::facility
            set prev_mnemonic       $::mnemonic
            set prev_severity       $::severity
            set prev_stream         $::stream
            set prev_traceback      $::traceback
            set prev_pid            $::pid
            set prev_process        $::process
            set prev_format_string  $::format_string
            set prev_msg_args       $::msg_args

            # construct a new syslog with the details of the login failure
            # alarm

            set ::timestamp  [cisco_service_timestamp]
            set ::facility "CC"
            set ::mnemonic "ALARM"
            set ::severity 6
            set ::stream 2
            set ::traceback "cc_internal_syslog"
            set ::pid ""
            set ::process ""
            set ::format_string ""
            set ::msg_args {}
            set ::orig_msg [format "%s %s: %s%s-%d-%s: %s" $::buginfseq $::timestamp "%"
$::facility $::severity $::mnemonic $ike_auth_fail_msg]

            # Send a syslog to be catched by the script that handles the
            # alarms
            esm_errmsg 0

            # restore all syslog global params

            set ::orig_msg        $prev_orig_msg
            set ::timestamp       $prev_timestamp
            set ::facility        $prev_facility
            set ::mnemonic        $prev_mnemonic
            set ::severity        $prev_severity
            set ::stream          $prev_stream
            set ::traceback       $prev_traceback
            set ::pid             $prev_pid
            set ::process         $prev_process
            set ::format_string   $prev_format_string
            set ::msg_args        $prev_msg_args
        }
    }

    proc process_ike_syslog {} {
        variable host_failed_ike_auth
        variable msg_to_watch

        # empty msg?
        if { [string length $::orig_msg] == 0} {
            return ""
        }
        set category "$::facility-$::severity-$::mnemonic"

        # Should I process this syslog?
        set need_to_process [query_category $category]
```

```
        if { $need_to_process == 0 } {
            return $::orig_msg
        }

        # Extract isakmp mode of the failed negotiation
        if { $::mnemonic == "IKMP_MODE_FAILURE" } {
            set isakmp_mode [lindex $::msg_args 0]
            if { $isakmp_mode != "Main" && $isakmp_mode != "Aggressive" } {
                return $::orig_msg
            }
        }

        # Extract ip address of the failed authentication
        set ip_address [lindex $::msg_args $msg_to_watch($category)]
        if { [string length $ip_address] == 0 } {
            return $::orig_msg
        }

        # Get current time and add it to the given ip_address list
        set time [clock seconds]
        lappend host_failed_ike_auth($ip_address) $time

        process_ike_auth_failure $ip_address

        return $::orig_msg
    }

    array set msg_to_watch {
        CRYPTO-6-IKMP_AUTH_FAIL                   1
        CRYPTO-6-IKMP_MODE_FAILURE                1
        CRYPTO-4-IKE_DENY_SA_REQ                  1
        CRYPTO-6-IKMP_BAD_DOI_SA                  1
        CRYPTO-6-IKMP_CRYPT_FAILURE               0
        CRYPTO-6-IKMP_BAD_CERT_USE                0
        CRYPTO-5-IKMP_INVAL_CERT                  0
        CRYPTO-4-IKMP_NO_SA                       0
        CRYPTO-6-IKMP_NO_ID_CERT_DN_MATCH         0
        CRYPTO-6-IKMP_NO_ID_CERT_ADDR_MATCH       0
        CRYPTO-6-IKMP_NO_ID_CERT_FQDN_MATCH       0
        CRYPTO-6-IKMP_NO_ID_CERT_USER_FQDN_MATCH  0
        CRYPTO-6-IKMP_NOT_ENCRYPTED               0
        CRYPTO-6-IKMP_SA_NOT_AUTH                 0
        CRYPTO-4-IKMP_HASH_SIZE_EXCEEDED          0
    }

    # Process the message
    process_ike_syslog

} ;# end namespace ike_auth_failures
```

Example: Tcl Scripts for the IKEv1 Phase 2 Failure Catcher

```
namespace eval ::ike_quickmode_failures {
    # namespace variables
    array set failed_quickmode_peers_id {}

    # Should I process this message ?
    proc query_category {cat} {
        variable msg_to_watch
```

```
        if { [info exists msg_to_watch($cat)] } {
            return 1
        } else {
            return 0
        }
    }

    # handle ike quickmode failure for a given ip address
    proc process_ike_quickmode_failure {ipaddress} {
        variable  failed_quickmode_peers_id

        set current_time    [clock seconds]

        #default values, change if get as cli args
        set alarm_threshold  1
        set time_gap         0

        # Get the list timestamps of previous ike authentication failures
        # of this ipaddress
        set time_list $failed_quickmode_peers_id($ipaddress)
        set list_len [llength $time_list]

        # First arg (if exists) is alarm threshlod
        if { [info exists ::cli_args] == 1 } {
            set alarm_threshold [lindex $::cli_args 0]

            # Second arg (if exists) is time gap
            if { [llength $::cli_args] > 1 } {
                set time_gap [lindex $::cli_args 1]
            }
        }

        if { $time_gap != 0 } {
            set i 0
            # run though the list and keep only the timestamps which are still
            # in the given time gap from current time
            while  { $i < $list_len } {
                if {[expr $current_time - [lindex $time_list $i]] <= $time_gap } {
                    lappend new_time_list [lindex $time_list $i]
                }
                incr i
            }

            # update the timestamp list for the given ipaddress
            set failed_quickmode_peers_id($ipaddress) $new_time_list
        } else {
            set new_time_list $failed_quickmode_peers_id($ipaddress)
        }

        # does the updated timestamp list has more items than the threshold
        if { [llength $new_time_list] >= $alarm_threshold } {
            if { $time_gap != 0 } {
                # Need to send a new alarm.
                set ike_auth_fail_msg [format "Processing of quick mode failed with %s %d
times in %d seconds interval" $ipaddress [llength $new_time_list] [expr $current_time -
[lindex $new_time_list 0]]]
            } else {
                set ike_auth_fail_msg [format "Processing of quick mode failed with %s %d
times" $ipaddress [llength $new_time_list]]
            }
            # store all current syslog global params
            set prev_orig_msg       $::orig_msg
            set prev_timestamp      $::timestamp
            set prev_facility       $::facility
```

```
                set prev_mnemonic       $::mnemonic
                set prev_severity       $::severity
                set prev_stream         $::stream
                set prev_traceback      $::traceback
                set prev_pid            $::pid
                set prev_process        $::process
                set prev_format_string  $::format_string
                set prev_msg_args       $::msg_args

                # construct a new syslog with the details of the login failure
                # alarm

                set ::timestamp  [cisco_service_timestamp]
                set ::facility "CC"
                set ::mnemonic "ALARM"
                set ::severity 6
                set ::stream 2
                set ::traceback "cc_internal_syslog"
                set ::pid ""
                set ::process ""
                set ::format_string ""
                set ::msg_args {}
                set ::orig_msg [format "%s %s: %s%s-%d-%s: %s" $::buginfseq $::timestamp "%"
$::facility $::severity $::mnemonic $ike_auth_fail_msg]

                # Send a syslog to be catched by the script that handles the
                # alarms
                esm_errmsg 0

                # restore all syslog global params

                set ::orig_msg        $prev_orig_msg
                set ::timestamp       $prev_timestamp
                set ::facility        $prev_facility
                set ::mnemonic        $prev_mnemonic
                set ::severity        $prev_severity
                set ::stream          $prev_stream
                set ::traceback       $prev_traceback
                set ::pid             $prev_pid
                set ::process         $prev_process
                set ::format_string   $prev_format_string
                set ::msg_args        $prev_msg_args
            }
        }

        proc process_ike_quickmode_syslog {} {
            variable failed_quickmode_peers_id
            variable msg_to_watch

            # empty msg?
            if { [string length $::orig_msg] == 0} {
                return ""
            }
            set category "$::facility-$::severity-$::mnemonic"

            # Should I process this syslog?
            set need_to_process [query_category $category]
            if { $need_to_process == 0 } {
                return $::orig_msg
            }

            # Extract isakmp mode of the failed negotiation
            set isakmp_mode [lindex $::msg_args 0]
            if {$::mnemonic == "IKMP_MODE_FAILURE" && $isakmp_mode != "Quick"} {
```

```
            return $::orig_msg
        }

        # Extract ip address of the peer who failed quickmode
        set ip_address [lindex $::msg_args $msg_to_watch($category)]
        if { [string length $ip_address] == 0} {
            return $::orig_msg
        }

        # Get current time and add it to the given ip_address list
        set time [clock seconds]
        lappend failed_quickmode_peers_id($ip_address) $time

        process_ike_quickmode_failure $ip_address

        return $::orig_msg
    }

    array set msg_to_watch {
        CRYPTO-6-IKMP_MODE_FAILURE   1
        CRYPTO-6-IKMP_SA_NOT_OFFERED 0
        CRYPTO-6-IPSEC_TRANSFORM_NOT_SUPPORTED 0
    }

    # Process the message
    process_ike_quickmode_syslog

} ;# end namespace ike_quickmode_failures
```

Example: Tcl Scripts for User Login Failures

```
namespace eval ::login_failure {
    # namespace variables
    array set user_failed_logins {}
    array set init_variables {}
    array set global_variables {}

    # Should I process this message ?
    proc query_category {cat} {
        variable msg_to_watch

        if { [info exists msg_to_watch($cat)] } {
            return $msg_to_watch($cat)
        } else {
            return 0
        }
    }

    proc close_all_vty {} {

        #get all the vty IDs
        set vty_list [exec show ru | inc line vty]

               # split the contents on newlines
        set list_of_lines [split $vty_list "\n"]

        set first_vty -1
        set last_vty -1

        # loop through the lines
        foreach line $list_of_lines {
```

```
            set vty_location [lsearch -exact $line vty]
            if {$vty_location != -1} {
                set vty_id [lindex $line [expr $vty_location + 1]]
                if {$first_vty == -1} {
                    set first_vty $vty_id
                }
                set last_vty $vty_id
            }
        }

                if {$first_vty == $last_vty} {
                        ios_config "line vty $first_vty" "transport input none"
                } else {
                        ios_config "line vty $first_vty $last_vty" "transport input none"
                }

        # go over all the ssh connections and close them - one after the other -
        set ssh_list [exec show ssh | in IN]

        # split the contents on newlines
        set list_of_lines [split $ssh_list "\n"]

        # loop through the lines
        foreach line $list_of_lines {
            set key_word [lsearch -exact $line IN]
            if {$key_word != -1} {
                set ssh_id [lindex $line 0]
                exec disconnect ssh $ssh_id
            }
        }
    }

    proc generate_alarm {syslog_msg mnemonic_msg} {

    # store all current syslog global params
         set prev_orig_msg       $::orig_msg
         set prev_timestamp      $::timestamp
         set prev_facility       $::facility
         set prev_mnemonic       $::mnemonic
         set prev_severity       $::severity
         set prev_stream         $::stream
         set prev_traceback      $::traceback
         set prev_pid            $::pid
         set prev_process        $::process
         set prev_format_string  $::format_string
         set prev_msg_args       $::msg_args

         # construct a new syslog with the details of the login failure
         # alarm

         set ::timestamp  [cisco_service_timestamp]
         set ::facility "CC"
         set ::mnemonic $mnemonic_msg
         set ::severity 6
         set ::stream 2
         set ::traceback "cc_internal_syslog"
         set ::pid ""
         set ::process ""
         set ::format_string ""
         set ::msg_args {}
         set ::orig_msg [format "%s %s: %s%s-%d-%s: %s" $::buginfseq $::timestamp "%"
$::facility $::severity $::mnemonic $syslog_msg]

         # Send a syslog to be catched by the script that handles the
```

```
        # alarms
        esm_errmsg 0

        # restore all syslog global params

        set ::orig_msg        $prev_orig_msg
        set ::timestamp       $prev_timestamp
        set ::facility        $prev_facility
        set ::mnemonic        $prev_mnemonic
        set ::severity        $prev_severity
        set ::stream          $prev_stream
        set ::traceback       $prev_traceback
        set ::pid             $prev_pid
        set ::process         $prev_process
        set ::format_string   $prev_format_string
        set ::msg_args        $prev_msg_args

    }

    # handle login failure for a given user
    proc process_login_failure {user} {
        variable  user_failed_logins
        variable  prev_orig_msg
       variable  global_variables

        set current_time    [clock seconds]

        #default values, change if get as cli args
        set alarm_threshold  1
        set time_gap         0

        # Get the list timestamps of previous login failures of this user
        set time_list $user_failed_logins($user)
        set list_len [llength $time_list]

       set time_gap $global_variables("time_gap")
       set alarm_threshold $global_variables("alarm_threshold")
       set total_remain_fails $global_variables("remain_fails")
       set max_allow_fails $global_variables("max_fails")

        if { $max_allow_fails != 0 } {
           set total_remain_fails [expr $total_remain_fails - 1]
           set global_variables("remain_fails") $total_remain_fails

           if { $total_remain_fails == 0 } {
               set global_variables("remain_fails") $max_allow_fails
               set login_fail_msg "Total number of Login failure ($max_allow_fails) was
exceeded, shutting down all VTYs"
               generate_alarm $login_fail_msg "INFO"
               close_all_vty
           }
       }

        if { $time_gap != 0 } {
            set i 0
            # run though the list and keep only the timestamps which are still
            # in the given time gap from current time
            while  { $i < $list_len } {
                if {[expr $current_time - [lindex $time_list $i]] <= $time_gap } {
                    lappend new_time_list [lindex $time_list $i]
                }
                incr i
            }
            # update the timestamp list for the given user
```

```
            set user_failed_logins($user) $new_time_list
        } else {
            set new_time_list $user_failed_logins($user)
        }

        # does the updated timestamp list has more items than the threshold
        if { [llength $new_time_list] >= $alarm_threshold } {
            if { $time_gap != 0 } {
                # Need to send a new login failure alarm.
                set login_fail_msg [format "%s for user %s %d times in %d seconds
interval" [lindex $::msg_args 3] [lindex $::msg_args 0] [llength $new_time_list] [expr
$current_time - [lindex $new_time_list 0]]]
            } else {
                set login_fail_msg [format "%s for user %s %d times" [lindex $::msg_args
3] [lindex $::msg_args 0] [llength $new_time_list]]
            }
           generate_alarm $login_fail_msg "ALARM"
        }
    }

    proc process_syslog {} {
        variable  user_failed_logins

        # empty msg?
        if { [string length $::orig_msg] == 0} {
            return ""
        }

        set category "$::facility-$::severity-$::mnemonic"

        # Should I process this syslog?
        set need_to_process [query_category $category]
        if { $need_to_process == 0 } {
            return $::orig_msg
        }

        # Extract username of the failed login try
        set username "[lindex $::msg_args 0]"
        if { [string length $username] == 0} {
            return $::orig_msg
        }

        # Get current time and add it to the given user list
        set time [clock seconds]
        lappend user_failed_logins($username) $time

        process_login_failure $username
        return $::orig_msg
    }

    array set msg_to_watch {
        SEC_LOGIN-4-LOGIN_FAILED 1
    }

    if { [array size init_variables] == 0 } {
        set init_variables("vars") "init"
        set global_variables("alarm_threshold") "0"
        set global_variables("time_gap") "0"
        set global_variables("max_fails") "0"
        set global_variables("remain_fails") "0"

        # First arg (if exists) is alarm threshlod
        if { [info exists ::cli_args] == 1 } {
            set global_variables("alarm_threshold") [lindex $::cli_args 0]
```

```
            # Second arg (if exists) is time gap
            if { [llength $::cli_args] > 1 } {
                set global_variables("time_gap") [lindex $::cli_args 1]
            }
           # Second arg (if exists) is time gap
           if { [llength $::cli_args] > 2 } {
               set global_variables("max_fails") [lindex $::cli_args 2]
               set global_variables("remain_fails") [lindex $::cli_args 2]
           }
        }
    }

    # Process the message
    process_syslog

} ;# end namespace login_failure
```

Example: Tcl Scripts for Information Flow Violations

```
namespace eval ::monitor_violation {

    # Should I process this message ?
    proc query_category {cat} {
        variable msg_to_watch

        if { [info exists msg_to_watch($cat)] } {
            return $msg_to_watch($cat)
        } else {
            return 0
        }
    }

    proc process_monitor_violation {newMsg} {
        # Need to send a new login failure alarm.
        set monitor_violation_msg $newMsg
        # store all current syslog global params
        set prev_orig_msg       $::orig_msg
        set prev_timestamp      $::timestamp
        set prev_facility       $::facility
        set prev_mnemonic       $::mnemonic
        set prev_severity       $::severity
        set prev_stream         $::stream
        set prev_traceback      $::traceback
        set prev_pid            $::pid
        set prev_process        $::process
        set prev_format_string  $::format_string
        set prev_msg_args       $::msg_args

        # construct a new syslog with the details of the login failure
        # alarm

        set ::timestamp  [cisco_service_timestamp]
        set ::facility "CC"
        set ::mnemonic "ALARM"
        set ::severity 6
        set ::stream 2
        set ::traceback "cc_internal_syslog"
        set ::pid ""
        set ::process ""
```

```
        set ::format_string ""
        set ::msg_args {}
        set ::orig_msg [format "%s %s: %s%s-%d-%s: %s" $::buginfseq $::timestamp "%"
$::facility $::severity $::mnemonic $monitor_violation_msg]

        # Send a syslog to be catched by the script that handles the
        # alarms
        esm_errmsg 0

        # restore all syslog global params

        set ::orig_msg        $prev_orig_msg
        set ::timestamp       $prev_timestamp
        set ::facility        $prev_facility
        set ::mnemonic        $prev_mnemonic
        set ::severity        $prev_severity
        set ::stream          $prev_stream
        set ::traceback       $prev_traceback
        set ::pid             $prev_pid
        set ::process         $prev_process
        set ::format_string   $prev_format_string
        set ::msg_args        $prev_msg_args
    }


    proc process_syslog {} {

        # empty msg?
        if { [string length $::orig_msg] == 0} {
            return ""
        }

        set category "$::facility-$::severity-$::mnemonic"

        # Should I process this syslog?
        set need_to_process [query_category $category]
        if { $need_to_process == 0 } {
            return $::orig_msg
        }

        if { [llength $::msg_args] < 1} {
            return $::orig_msg
        }

       set list_of_args [split $::msg_args " "]

        # now check for MONITOR-6-VIOLATION inside the msg args
       set mon_event_str %MONITOR-6-VIOLATION:
       set vio_loc [lsearch -exact $list_of_args $mon_event_str ]

       if {$vio_loc != -1} {
           set arg1 [lindex $list_of_args [expr $vio_loc + 2]]
           set arg2 [lindex $list_of_args [expr $vio_loc + 4]]
           set arg3 [lindex $list_of_args [expr $vio_loc + 7]]
       } else {
            return $::orig_msg
       }

        set msg [format "Information Flow policy violation for ACL %s logged %d times in
%d seconds" $arg1 $arg2 $arg3]

       process_monitor_violation $msg
        return $::orig_msg
    }
```

```
    array set msg_to_watch {
        IOSXE-6-PLATFORM    1
    }

    # Process the message
    process_syslog

} ;# end namespace monitor_violation
```

Example: Tcl Scripts for VPN Events

```
namespace eval ::monitor_vpn_event {

    # Should I process this message ?
    proc query_category {cat} {
        variable msg_to_watch

        if { [info exists msg_to_watch($cat)] } {
            return $msg_to_watch($cat)
        } else {
            return 0
        }
    }

    proc process_monitor_vpn_event {newMsg} {
        set monitor_vpn_event_msg $newMsg

        # store all current syslog global params
        set prev_orig_msg       $::orig_msg
        set prev_timestamp      $::timestamp
        set prev_facility       $::facility
        set prev_mnemonic       $::mnemonic
        set prev_severity       $::severity
        set prev_stream         $::stream
        set prev_traceback      $::traceback
        set prev_pid            $::pid
        set prev_process        $::process
        set prev_format_string  $::format_string
        set prev_msg_args       $::msg_args

        # construct a new syslog with the details of the login failure
        # alarm

        set ::timestamp  [cisco_service_timestamp]
        set ::facility "CC"
        set ::mnemonic "ALARM"
        set ::severity 6
        set ::stream 2
        set ::traceback "cc_internal_syslog"
        set ::pid ""
        set ::process ""
        set ::format_string ""
        set ::msg_args {}
        set ::orig_msg [format "%s %s: %s%s-%d-%s: %s" $::buginfseq $::timestamp "%"
$::facility $::severity $::mnemonic $monitor_vpn_event_msg]

        # Send a syslog to be catched by the script that handles the
        # alarms
        esm_errmsg 0
```

```
        # restore all syslog global params

        set ::orig_msg        $prev_orig_msg
        set ::timestamp       $prev_timestamp
        set ::facility        $prev_facility
        set ::mnemonic        $prev_mnemonic
        set ::severity        $prev_severity
        set ::stream          $prev_stream
        set ::traceback       $prev_traceback
        set ::pid             $prev_pid
        set ::process         $prev_process
        set ::format_string   $prev_format_string
        set ::msg_args        $prev_msg_args
    }


    proc process_syslog {} {

        # empty msg?
        if { [string length $::orig_msg] == 0} {
            return ""
        }

        set category "$::facility-$::severity-$::mnemonic"

        # Should I process this syslog?
        set need_to_process [query_category $category]
        if { $need_to_process == 0 } {
            return $::orig_msg
        }

        if { [llength $::msg_args] < 1} {
            return $::orig_msg
        }

       set list_of_args [split $::msg_args " "]

       # now check for MONITOR-3-VPN_EVENT inside the msg args
       set vpn_event_str %MONITOR-3-VPN_EVENT:
       set vpn_loc [lsearch -exact $list_of_args $vpn_event_str ]

       if {$vpn_loc != -1} {
           set arg1 [lindex $list_of_args [expr $vpn_loc + 4]]
           set arg2 [lindex $list_of_args [expr $vpn_loc + 8]]
       } else {
            return $::orig_msg
       }

       set msg [format "Ipsec event type %s occured %d times" $arg1 $arg2]

        process_monitor_vpn_event $msg
        return $::orig_msg
    }

    array set msg_to_watch {
        IOSXE-3-PLATFORM 1
    }

    # Process the message
    process_syslog

} ;# end namespace monitor_vpn_event
```

Example: Tcl Scripts for Configuring vty Devices

```
namespace eval ::CC_vty_monitor {

    # Should I process this message ?
    proc query_category {cat} {
        variable msg_to_watch

        if { [info exists msg_to_watch($cat)] } {
            return $msg_to_watch($cat)
        } else {
            return 0
        }
    }

    proc process_syslog {} {

        # empty msg?
        if { [string length $::orig_msg] == 0} {
            return ""
        }

        set category "$::facility-$::severity-$::mnemonic"

        # Should I process this syslog?
        set need_to_process [query_category $category]
        if { $need_to_process == 0 } {
            return $::orig_msg
        }

       # got success login - so now conf the VTYs
       set users_output [exec show users wide | inc vty]

       if {[llength $users_output] <= 1} {
            return $::orig_msg
       }


       # split the contents on newlines
       set list_of_lines [split $users_output "\n"]

       set first_vty -1
       set last_vty -1

       # loop through the lines
       foreach line $list_of_lines {
           set vty_location [lsearch -exact $line vty]
           if {$vty_location != -1} {
               set vty_id [lindex $line [expr $vty_location + 1]]
               if {$first_vty == -1} {
                   set first_vty $vty_id
               }
               set last_vty $vty_id
           }
       }
       if {$first_vty == $last_vty} {
           ios_config "line vty $first_vty" "monitor"
       } else {
           ios_config "line vty $first_vty $last_vty" "monitor"
       }

       return $::orig_msg
    }
```

```
    array set msg_to_watch {
        SEC_LOGIN-5-LOGIN_SUCCESS 1
    }

    # Process the message
    process_syslog
} ;
# end CC_vty_monitor
```

Example: Tcl Scripts for Periodic FIPS

```
namespace eval ::CC_periodic_fips {
    array set fips_periodic_started {}

    proc fips_periodic_run {} {
        if { [info exists ::CC_periodic_fips::fips_delta] } {
            set now [clock seconds]
            set tmp_val [expr $::CC_periodic_fips::curr_time +
$::CC_periodic_fips::fips_delta]
            if {$now > $tmp_val} {
                set ::CC_periodic_fips::curr_time $tmp_val
                exec "test crypto self-test"
            }
        }
    }
    # Initialize processes for alonTimer
    if { [array size fips_periodic_started] == 0 } {
        variable fips_periodic_started
        set fips_periodic_started("fips_periodic") "started"

        set curr_time [clock seconds]
        if { [info exists ::cli_args] } {
            set fips_delta $::cli_args
        } else {
            puts "bad cli argument, configure the script again"
        }
    }
    ::CC_periodic_fips::fips_periodic_run

    # just pass the message to next filter
    return $::orig_msg
} ;
# end CC_periodic_fips
```

Example: Tcl Scripts for the IPsec Policy Violation Category Watcher

```
namespace eval ::Ipsec_monitor {
    array set ipsec_monitor_started {}

    array set qfp_options {
        active  1
        standby 2
    }
```

```
array set type_options {
    decrypt-failed  1
    encrypt-failed  2
    replay          3
}

proc query_qfp {cat} {
    variable qfp_options

    if { [info exists qfp_options($cat)] } {
        return $qfp_options($cat)
    } else {
        return 0
    }
}

proc query_type {cat} {
    variable type_options

    if { [info exists type_options($cat)] } {
        return $type_options($cat)
    } else {
        return 0
    }
}

   proc generate_alarm {syslog_msg mnemonic_msg} {

# store all current syslog global params
    set prev_orig_msg       $::orig_msg
    set prev_timestamp      $::timestamp
    set prev_facility       $::facility
    set prev_mnemonic       $::mnemonic
    set prev_severity       $::severity
    set prev_stream         $::stream
    set prev_traceback      $::traceback
    set prev_pid            $::pid
    set prev_process        $::process
    set prev_format_string  $::format_string
    set prev_msg_args       $::msg_args

    # construct a new syslog with the details of the login failure
    # alarm

    set ::timestamp  [cisco_service_timestamp]
    set ::facility "CC"
    set ::mnemonic $mnemonic_msg
    set ::severity 6
    set ::stream 2
    set ::traceback "cc_internal_syslog"
    set ::pid ""
    set ::process ""
    set ::format_string ""
    set ::msg_args {}
    set ::orig_msg [format "%s %s: %s%s-%d-%s: %s" $::buginfseq $::timestamp "%"
$::facility $::severity $::mnemonic $syslog_msg]

    # Send a syslog to be catched by the script that handles the
    # alarms
    esm_errmsg 0

    # restore all syslog global params
```

```
            set ::orig_msg        $prev_orig_msg
            set ::timestamp       $prev_timestamp
            set ::facility        $prev_facility
            set ::mnemonic        $prev_mnemonic
            set ::severity        $prev_severity
            set ::stream          $prev_stream
            set ::traceback       $prev_traceback
            set ::pid             $prev_pid
            set ::process         $prev_process
            set ::format_string   $prev_format_string
            set ::msg_args        $prev_msg_args

        }


        # check if it was already excuted, if not read CLI args
        # verify that they are correct and configure the required command

        if { [array size ipsec_monitor_started] == 0 } {
            variable ipsec_monitor_started
            set ipsec_monitor_started("Ipsec") "started"

            # check if all the args are here
            if { [info exists ::cli_args] == 1 } {

                # check if all 3 args are here
                if { [llength $::cli_args] > 2 } {
                    set qfp   [lindex $::cli_args 0]
                    set type  [lindex $::cli_args 1]
                    set count [lindex $::cli_args 2]

                    set check_input [query_qfp $qfp]
                    if { $check_input == 0 } {
                        generate_alarm "Error: bad arg ($qfp)" "INFO"
                        return $::orig_msg
                    }

                    set check_input [query_type $type]
                    if { $check_input == 0 } {
                        generate_alarm "Error: bad arg ($type)" "INFO"
                        return $::orig_msg
                    }

                    if {$count < 0} {
                        generate_alarm "Error: bad arg ($count)" "INFO"
                        return $::orig_msg
                    }

                    exec set platform hardware qfp $qfp feature ipsec event-monitor type $type
count $count
                }

            }

            return $::orig_msg

        }

        # just pass the message to next filter
        return $::orig_msg
} ;
# end namespace Ipsec_monitor
```

Example: Tcl Scripts for the Exclude Syslog Messages with Keywords

```
namespace eval ::syslog_exclude {
    array set msg_to_confirm {
        PARSER-5-CFGLOG_LOGGEDCMD  1
    }

    proc query_confirm {cat} {
        variable msg_to_confirm

        if { [info exists msg_to_confirm($cat)] } {
            return $msg_to_confirm($cat)
        } else {
            return 0
        }
    }

    proc process_syslog {} {
        # empty msg?
        if { [string length $::orig_msg] == 0} {
            return ""
        }

        if { [info exists ::cli_args] == 0 } {
            return $::orig_msg
        }

        if { $::traceback == "cc_internal_syslog"} {
            # This is common creteria internal syslog
            return $::orig_msg
        }
        set category "$::facility-$::severity-$::mnemonic"
        # Is this a confirmation syslog?
        set need_to_confirm [query_confirm $category]
        if { $need_to_confirm == 1 } {
            if { "event manager environment confirm_alarm" == [lrange [split [lindex
$::msg_args 1]] 0 3]} {
                # This is common creteria internal syslog
                set ::stream  2
                return $::orig_msg
            }
        }
        set args_count [llength $::cli_args]
        set i 0
        while { $i < $args_count} {
            set result [regexp [lindex $::cli_args $i] $::orig_msg]
            if { $result == 1} {
                # found token in $::orig_msg
                return ""
            }
            incr i
        }
        return $::orig_msg
    }

    # Process the message
    process_syslog

} ;# end namespace syslog_exclude
```

Example: Tcl Scripts for the Include Syslog Messages with Keywords

```
namespace eval ::syslog_include {
    array set msg_to_confirm {
        PARSER-5-CFGLOG_LOGGEDCMD  1
    }

    proc query_confirm {cat} {
        variable msg_to_confirm

        if { [info exists msg_to_confirm($cat)] } {
            return $msg_to_confirm($cat)
        } else {
            return 0
        }
    }

    proc process_syslog {} {
        # empty msg?
        if { [string length $::orig_msg] == 0} {
            return ""
        }

        if { [info exists ::cli_args] == 0 } {
            return $::orig_msg
        }

        if {  $::traceback == "cc_internal_syslog"} {
            # This is common creteria internal syslog
            return $::orig_msg
        }
        set category "$::facility-$::severity-$::mnemonic"
        # Is this a confirmation syslog?
        set need_to_confirm [query_confirm $category]
        if { $need_to_confirm == 1 } {
            if { "event manager environment confirm_alarm" == [lrange [split [lindex
$::msg_args 1]] 0 3]} {
                # This is common creteria internal syslog
                set ::stream  2
                return $::orig_msg
            }
        }
        set args_count [llength $::cli_args]
        set i 0
        while { $i < $args_count} {
            set result [regexp [lindex $::cli_args $i] $::orig_msg]
            if { $result == 1} {
                # found token in $::orig_msg
                return $::orig_msg
            }
            incr i
        }
        return ""
    }

    # Process the message
    process_syslog

} ;# end namespace syslog_include
```

Example: Tcl Scripts for Timer Events

```
namespace eval ::Timer {
    array set timer_process_started {}

    proc timer_run {time_interval} {
        set curr_time [clock seconds]

        after $time_interval ::Timer::timer_run $time_interval

        set ::orig_msg "seconds $curr_time"
        set ::timestamp  [cisco_service_timestamp]
        set ::facility "CC"
        set ::mnemonic "TIMER"
        set ::severity 6
        set ::stream 2
        set ::traceback "cc_internal_syslog"
        set ::pid ""
        set ::process ""
        set ::format_string "seconds %d"
        set ::msg_args {$curr_time}
        esm_errmsg 0
    }

    # Initialize processes for Timer
    if { [array size timer_process_started] == 0 } {
        variable timer_process_started
        set timer_process_started("timer") "started"

        #default value 1 minute
        set time_interval 60000
        if { [info exists ::cli_args] == 1 } {
            set time_interval [expr [lindex $::cli_args 0] * 1000]
        }
        after 60000 ::Timer::timer_run $time_interval

       #set the debug flags we want - isakmp & ipsec
       exec debug crypto isakmp error
       exec debug crypto ipsec error
    }

    # just pass the message to next filter
    return $::orig_msg
} ;
# end namespace Timer
```

For More Information

The following sections provide references related to the Common Criteria Tcl Scripts feature.

Related Documents

Related Topic	Document Title
Embedded Syslog Manager	Embedded Syslog Manager module
Network Management Configurations	*Cisco IOS Network Management Configuration Guide*
Network Management commands (including Tcl and logging commands): complete command syntax, defaults, command mode, command history, usage guidelines, and examples	*Cisco IOS Network Management Command Reference*