Python Cache Support for ESM Applications

In This Chapter

This section provides information about Python cache support for ESM applications.

Topics in this section include:

Applicability

This feature is applicable to 7750 SR-7/12/12e systems, and 7450 systems in mixed mode, with CPM3 or later. It is also applicable to the virtualized simulator but only when running as a distributed simulator. It is not applicable to the 7750 SR c4/12.

The configuration was tested with SR OS release 12.0.R4.

Overview

SR OS sports an embedded Python scripting engine which can be used to manipulate selected messages of protocols including DHCPv4, DHCPv6, RADIUS and Diameter.

Python cache provides a central key-value memory cache with a set of APIs allowing Python scripts to store and retrieve strings across different runs of the same or even different Python scripts.

The following are some basic concepts of Python cache:

- Multiple Python policies can be defined, and each Python policy has a separate cache; only scripts configured in the Python policy can access and share that policy's cache.
- A cache entry consists of a key and a value, both of them being strings. The key is used to search and fetch the cache entry.
- A cache entry has a lifetime and the system automatically removes expired entries.

The following are the Python cache APIs:

- alc.cache.save(val,key): Save the value identified by the key into the cache.
- alc.cache.retrieve(key): Return the cached entry's value identified by the key.
- alc.cache.clear(key): Remove the entry identified by the key from the cache.
- cache.get_lifetime(key): The system returns an integer indicating the remaining lifetime of the specified entry expressed in seconds.
- cache.set_lifetime(key,new_lifetime): new_lifetime is an integer; the system sets the remaining lifetime of the specified entry (in seconds).

Configuration

The test topology is shown in Figure 397.

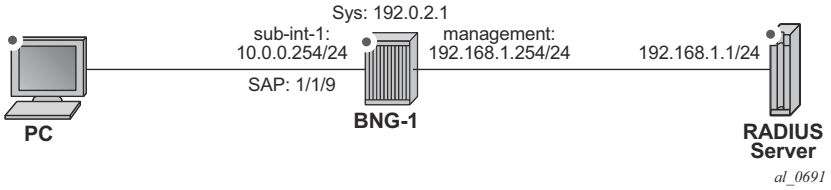

Figure 397: Test Topology

The example shows how the Python cache can be used to store multiple class attributes from the RADIUS access-accept packets and reflect them into RADIUS accounting request packets (start/interim-update/stop).

Test setup:

- A PC is used as DHCPv4 host, connect to SAP 1/1/9 of BNG-1
- SAP 1/1/9 is attached to group interface grp-int-1, which is under subscriber interface sub-int-1 of IES 1
- DHCP host is authenticated via RADIUS server, which resides in management routing instance of BNG-1
- A Python script python-script-1 stores all class attributes in access-accept and add stored attributes into RADIUS accounting requests.
- RADIUS user-name is used as cache key, which is the MAC address of the PC.

The Python cache configuration commands are shown below.

```
config>python>py-policy>
     [no] cache [create]
           [no] entry-size <size>
           [no] max-entries <count>
           [no] max-entry-lifetime [days <days>] [hrs <hours>]
                                   [min <minutes>] [sec <seconds>]
           [no] mcs-peer <ip-address> sync-tag <[32 chars max]>
           [no] minimum-lifetimes
               [no] high-availability <seconds>
               [no] multi-chassis-redundancy <seconds>
               [no] persistence <seconds>
           [no] persistence
           [no] shutdown
```

```
config>system>persistence>
    python-policy-cache
        [no] description <desc>
        [no] location <cflash-id>

config>redundancy>multi-chassis>peer>sync>
    [no] python
```

Refer to the SR OS Triple Play Guide for details of above commands. The basic configuration of the Python cache is the **cache create** statement in the python-policy as shown below:

```
config>python>py-policy>
-----------------------------------------------
            cache create
                no shutdown
            exit
            radius access-accept direction ingress script "python-script-1"
            radius accounting-request direction egress script "python-script-1"
-----------------------------------------------
```

The **cache create** configuration enables the cache support for the Python policy. The system's behavior can be tuned in the following aspects:

- Configure **entry-size** and **max-entries** to limit the memory usage.
- Configure **max-entry-lifetime** to specify the maximum lifetime.
- Enable **persistence** to make cache entries persistent across reboot.
- Configure **mcs-peer** to enable Multi-Chassis Synchronization.

In this example only the basic cache configuration is used.

Step 0. Configuring ESM

As a prerequisite ESM must be enabled and as such BNG-1 is configured as follows:

- An **authentication-policy radius-auth-policy-1** is used to authenticate DHCPv4 hosts on group interface grp-int-1.
- A **radius-accounting-policy radius-acct-policy-1** is configured in the **sub-profile sub-profile-1** to enable RADIUS accounting.
- The **user-name** is included in **radius-acct-policy-1** since the User-Name is used as cache entry key.
- Interim-update is enabled in the **radius-acct-policy-1** with an interval 5 minutes.
- The local DHCPv4 server **dhcpv4-svr** is used to assign address to hosts.

```
#--------------------------------------------------
echo "Management Router Configuration"
#--------------------------------------------------
    router management
        radius-server
            server "radius-svr-1" address 192.168.1.1 secret
                                  "iaCuILBunKirJurE4jK2URAnzip6nK32" hash2 create
            exit
        exit
    exit
#--------------------------------------------------
echo "Router (Network Side) Configuration"
#--------------------------------------------------
    router
        dhcp
            local-dhcp-server "dhcpv4-svr" create
            exit
        exit
        interface "system"
            address 192.0.2.1/32
            local-dhcp-server "dhcpv4-svr"
            no shutdown
        exit
    exit

#--------------------------------------------------
echo "Subscriber-mgmt Configuration"
#--------------------------------------------------
    subscriber-mgmt
        authentication-policy "radius-auth-policy-1" create
            password "mcgLj0q0695Dp.pD5DthrCv9Bu8X2qPVSvGYWQmCgUg" hash2
            radius-server-policy "radius-svr-policy-1"
        exit
        radius-accounting-policy "radius-acct-policy-1" create
            update-interval 5
            update-interval-jitter absolute 0
            include-radius-attribute
```

```
                user-name
            exit
            radius-server-policy "radius-svr-policy-1"
        exit
        sla-profile "sla-profile-1" create
        exit
        sub-profile "sub-profile-1" create
            radius-accounting-policy "radius-acct-policy-1"
        exit
    exit
#--------------------------------------------------
echo "Service Configuration"
#--------------------------------------------------
    service
        ies 1 customer 1 create
            subscriber-interface "sub-int-1" create
                address 10.0.0.254/24
                group-interface "grp-int-1" create
                    dhcp
                        server 192.0.2.1
                        gi-address 10.0.0.254
                        no shutdown
                    exit
                    authentication-policy "radius-auth-policy-1"
                    sap 1/1/9 create
                        sub-sla-mgmt
                            def-sub-id use-auto-id
                            def-sub-profile "sub-profile-1"
                            def-sla-profile "sla-profile-1"
                            no shutdown
                        exit
                    exit
                exit
            exit
            no shutdown
        exit
    exit
#--------------------------------------------------
echo "Local DHCP Server (Base Router) Configuration"
#--------------------------------------------------
    router
        dhcp
            local-dhcp-server "dhcpv4-svr" create
                use-gi-address
                pool "addr-pool-1" create
                    subnet 10.0.0.0/24 create
                        options
                            subnet-mask 255.255.255.0
                            default-router 10.0.0.254
                        exit
                        address-range 10.0.0.1 10.0.0.100
                    exit
                exit
                no shutdown
            exit
        exit
    exit
#--------------------------------------------------
echo "AAA Configuration"
```

```
#--------------------------------------------------
    aaa
        radius-server-policy "radius-svr-policy-1" create
            servers
                router "management"
                server 1 name "radius-svr-1"
            exit
        exit
    exit
```

Step 1. Create the Python script file.

A Python script is created and stored on the local storage, for example as CF3:\python_cache.py. This script handles the RADIUS packets listed below:

- Access-Accept — All class attributes from the access-accept packets are stored in the cache by combining them into a single string. The user-name (MAC address) is used as the key, and the format of this string is:
  - 1st byte is the number of class attributes in this string
  - 2nd – nth bytes: each byte holds the number of bytes for class-n attributes
  - Rest of bytes: combined string of class-1 to class-n
- Acct-Start — Retrieves the stored combined class string using the user name as key. The combined string is parsed and split into the individual class attributes and then inserted into the packet.
- Interim-Update — The cached entry is parsed and the stored class attributes are inserted, then the lifetime is reset to 15 minutes. This is greater than the interim-update interval (5 minutes) so the cached entry will not expire before next interim-update arrives.
- Acct-Stop — The cached entry is parsed and the stored class attributes are inserted, then the cached entry is removed.

Note: There is some error checking and exception handling logic in the script which causes the script to drop the packet if certain errors occur. This script is only an overview example so the error handling logic should be added according to real application requirements. In addition to the exception handling logic included in the script, the **action-on-fail** command could be used in the **python-script** command to define system's action upon failed execution, for example when an un-captured exception is encountered.

```
#------------------------------------------------------------------------------
# Name:       Alcatel-Lucent SR OS Python Cache Support Example
# Purpose:    This script is used to demonstrate SR OS python cache support for
#               the following use case:
#                  RADIUS sever returns multiple Class attributes in
#                  access-accept, these Class attributes need to be reflected
#                  in accounting request packets
#------------------------------------------------------------------------------
from alc import cache
from alc import radius
import struct
def main():
    radius_header = radius.header()
    if radius_header['code'] == 2: # in case of access-accept
        entry_key = radius.attributes.get(1) # use user-name as the cache entry
                                             # key
        if entry_key == "": #drop the packet if there is no user-name present
            radius.drop()
            return
        class_list = radius.attributes.getTuple(25) # get a list of Class
```

```
                                                    # attributes
            if class_list == (): #drop the packet if there is no class present
                radius.drop()
                return
            class_len_str = ''
            class_str = ''
            class_count = 0
            for radius_class in class_list:
                class_len_str += chr(len(radius_class))
                class_str += radius_class
                class_count += 1
            entry_val = chr(class_count)+class_len_str+class_str
            try:
                cache.save(entry_val,entry_key)
            except:
                radius.drop() #drop the packet if cache.save fails
                return

        elif radius_header['code'] == 4: # in case of acct-request
            entry_key = radius.attributes.get(1)
            if entry_key == "": #drop the packet if there is no user-name present
                radius.drop()
                return
            try:
                entry_val = cache.retrieve(entry_key)
            except:#drop the packet if cache.retrieve fails
                radius.drop()
                return
            class_count = ord(entry_val[0])
            pos = class_count+1
            class_list = []
            for i in range(class_count):
                class_len = ord(entry_val[i+1])
                class_list.append(entry_val[pos:pos+class_len])
                pos += class_len
            radius.attributes.set(25,tuple(class_list))
            acct_type=struct.unpack('>I',radius.attributes.get(40))[0]
            if acct_type == 2: # in case of acct-stop
                cache.clear(entry_key) # remove the cache entry
            elif acct_type == 3: # in case of interim-update
                try:
                    cache.set_lifetime(entry_key,900) # reset the lifetime
                except:#drop the packet if cache.set_lifetime fails
                    radius.drop()
                    return

main()
```

Step 2. Configure **python-script** and **python-policy**.

- Enable **python-script-1** to process received (ingress) access-accept packets and transmitted (egress) accounting-request packets.
- Enable the Python cache by configuring **cache create** in **python-policy-1.**
- Reference **python-policy-1** in the **radius-server-policy-1**.

```
#--------------------------------------------------
echo "PYTHON Configuration"
#--------------------------------------------------
    python
        python-script "python-script-1" create
            primary-url "cf3:/python_cache.py"
            no shutdown
        exit
        python-policy "python-policy-1" create
            cache create
                no shutdown
            exit
            radius access-accept direction ingress script "python-script-1"
            radius accounting-request direction egress script "python-script-1"
        exit
exit
#--------------------------------------------------
echo "AAA Configuration"
#--------------------------------------------------
    aaa
        radius-server-policy "radius-svr-policy-1" create
            python-policy "python-policy-1"
```

Step 3. Configure the RADIUS server.

- Configure the RADIUS server so that:
  - The DHCP host is authenticated via its MAC address.
  - The RADIUS server returns the following class attributes and values in access-accept packets:
    - "Class-1"
    - "Class-22"
    - "Class-333"

Step 4. Enable debug on BNG-1 to observe the Python cache in action.

- Enable the following debug on BNG-1:

```
debug
    router "Base"
        ip
            dhcp
                detail-level low
                mode egr-ingr-and-dropped
            exit
        exit
    exit
    router "management"
        radius
            packet-type authentication accounting coa
            detail-level medium
        exit
    exit
    python
        python-script "python-script-1"
            script-all-info
        exit
    exit
exit

A:BNG-1>config>log# info
----------------------------------------------
        log-id 10
            from debug-trace
            to session
        exit
----------------------------------------------
```

Step 5. Initiate DHCPv4 on the PC.

- Initiate the DHCPv4 process on the PC. When the PC sends out a DHCPv4 discover message, BNG-1 contacts the RADIUS server to authenticate the user and a DHCPv4 ESM host is created.
- RADIUS accounting-start will be sent upon host creation and interim-update will be sent every 5 minutes.

The following describes the debug output:

- The system initiates RADIUS authentication upon receipt of a DHCP discovery message.
- The RADIUS server returns an access-accept with the three class attributes.
- The system executes python-script-1, stores the three class attributes.
- After the DHCP host is created, the system sends RADIUS accounting-start and interim-update messages in which python-script-1 adds the three stored class attributes.

```
26 2014/06/26 03:02:18.34 UTC MINOR: DEBUG #2001 Base PIP
"PIP: DHCP
instance 1 (Base), interface index 3 (grp-int-1),
   received DHCP Boot Request on Interface grp-int-1 (1/1/9) Port 67

   H/W Type: Ethernet(10Mb)  H/W Address Length: 6
   ciaddr: 0.0.0.0           yiaddr: 0.0.0.0
   siaddr: 0.0.0.0           giaddr: 0.0.0.0
   chaddr: 00:20:fc:1e:cd:53    xid: 0x1e866b74
"

27 2014/06/26 03:02:18.34 UTC MINOR: DEBUG #2001 management RADIUS
"RADIUS: Transmit
  Access-Request(1) 192.168.1.1:1812 id 7 len 63 vrid 4095 pol radius-svr-policy -1
    USER NAME [1] 17 00:20:fc:1e:cd:53
    PASSWORD [2] 16 R/mc867ChKkdx50PJauB5U
    NAS IP ADDRESS [4] 4 192.168.1.254
"

28 2014/06/26 03:02:18.34 UTC MINOR: DEBUG #2001 management RADIUS
"RADIUS: Receive
  Access-Accept(2) id 7 len 69 from 192.168.1.1:1812 vrid 4095 pol radius-svr-po
licy-1
    CLASS [25] 7 0x436c6173732d31
    CLASS [25] 8 0x436c6173732d3232
    CLASS [25] 9 0x436c6173732d333333
    USER NAME [1] 17 00:20:fc:1e:cd:53
"

29 2014/06/26 03:02:18.34 UTC MINOR: DEBUG #2001 Base Python Output
"Python Output: python-script-1
"

30 2014/06/26 03:02:18.34 UTC MINOR: DEBUG #2001 Base Python Result
"Python Result: python-script-1
"
```

```
31 2014/06/26 03:02:18.34 UTC MINOR: DEBUG #2001 management RADIUS
"RADIUS: Script
Access-Accept(2) id 7 len 69 from 192.168.1.1:1812 policy python-policy-1 stat us success
"

32 2014/06/26 03:02:18.34 UTC MINOR: DEBUG #2001 Base PIP
"PIP: DHCP
instance 1 (Base),
   transmitted DHCP Boot Request to 192.0.2.1 Port 67

   H/W Type: Ethernet(10Mb)  H/W Address Length: 6
   ciaddr: 0.0.0.0           yiaddr: 0.0.0.0
   siaddr: 0.0.0.0           giaddr: 10.0.0.254
   chaddr: 00:20:fc:1e:cd:53    xid: 0x1e866b74
"

33 2014/06/26 03:02:18.34 UTC MINOR: DEBUG #2001 Base PIP
"PIP: DHCP
instance 1 (Base),
   received DHCP Boot Reply on 192.0.2.1 Port 67

   H/W Type: Ethernet(10Mb)  H/W Address Length: 6
   ciaddr: 0.0.0.0           yiaddr: 10.0.0.1
   siaddr: 192.0.2.1         giaddr: 10.0.0.254
   chaddr: 00:20:fc:1e:cd:53    xid: 0x1e866b74
"

34 2014/06/26 03:02:18.34 UTC MINOR: DEBUG #2001 Base PIP
"PIP: DHCP
instance 1 (Base), interface index 3 (grp-int-1),
   transmitted DHCP Boot Reply to Interface grp-int-1 (1/1/9) Port 68

   H/W Type: Ethernet(10Mb)  H/W Address Length: 6
   ciaddr: 0.0.0.0           yiaddr: 10.0.0.1
   siaddr: 192.0.2.1         giaddr: 10.0.0.254
   chaddr: 00:20:fc:1e:cd:53    xid: 0x1e866b74
"

35 2014/06/26 03:02:18.34 UTC MINOR: DEBUG #2001 Base PIP
"PIP: DHCP
instance 1 (Base), interface index 3 (grp-int-1),
   received DHCP Boot Request on Interface grp-int-1 (1/1/9) Port 67

   H/W Type: Ethernet(10Mb)  H/W Address Length: 6
   ciaddr: 0.0.0.0           yiaddr: 0.0.0.0
   siaddr: 0.0.0.0           giaddr: 0.0.0.0
   chaddr: 00:20:fc:1e:cd:53    xid: 0x1e866b74
"

36 2014/06/26 03:02:18.35 UTC MINOR: DEBUG #2001 Base PIP
"PIP: DHCP
instance 1 (Base),
   transmitted DHCP Boot Request to 192.0.2.1 Port 67

   H/W Type: Ethernet(10Mb)  H/W Address Length: 6
   ciaddr: 0.0.0.0           yiaddr: 0.0.0.0
   siaddr: 0.0.0.0           giaddr: 10.0.0.254
   chaddr: 00:20:fc:1e:cd:53    xid: 0x1e866b74
"
```

```
37 2014/06/26 03:02:18.35 UTC MINOR: DEBUG #2001 Base PIP
"PIP: DHCP
instance 1 (Base),
   received DHCP Boot Reply on 192.0.2.1 Port 67

   H/W Type: Ethernet(10Mb)  H/W Address Length: 6
   ciaddr: 0.0.0.0           yiaddr: 10.0.0.1
   siaddr: 192.0.2.1         giaddr: 10.0.0.254
   chaddr: 00:20:fc:1e:cd:53    xid: 0x1e866b74
"

38 2014/06/26 03:02:18.38 UTC MINOR: DEBUG #2001 Base PIP
"PIP: DHCP
instance 1 (Base), interface index 3 (grp-int-1),
   transmitted DHCP Boot Reply to Interface grp-int-1 (1/1/9) Port 68

   H/W Type: Ethernet(10Mb)  H/W Address Length: 6
   ciaddr: 0.0.0.0           yiaddr: 10.0.0.1
   siaddr: 192.0.2.1         giaddr: 10.0.0.254
   chaddr: 00:20:fc:1e:cd:53    xid: 0x1e866b74
"

39 2014/06/26 03:02:18.38 UTC MINOR: DEBUG #2001 Base Python Output
"Python Output: python-script-1
"

40 2014/06/26 03:02:18.38 UTC MINOR: DEBUG #2001 Base Python Result
"Python Result: python-script-1
RADIUS Attribute: Type 25, SET
     'Class-1'
     'Class-22'
     'Class-333'
"

41 2014/06/26 03:02:18.38 UTC MINOR: DEBUG #2001 management RADIUS
"RADIUS: Transmit
  Accounting-Request(4) 192.168.1.1:1813 id 8 len 152 vrid 4095 pol radius-svr-policy-1
    STATUS TYPE [40] 4 Start(1)
    NAS IP ADDRESS [4] 4 192.168.1.254
    USER NAME [1] 17 00:20:fc:1e:cd:53
    SESSION ID [44] 63 00:20:fc:1e:cd:53|1/1/9@1/1/9@sla-profile-1_2014/06/26 03
:02:18
    EVENT TIMESTAMP [55] 4 1403751738
    CLASS [25] 7 0x436c6173732d31
    CLASS [25] 8 0x436c6173732d3232
    CLASS [25] 9 0x436c6173732d333333
"

42 2014/06/26 03:02:18.38 UTC MINOR: DEBUG #2001 management RADIUS
"RADIUS: Receive
  Accounting-Response(5) id 8 len 20 from 192.168.1.1:1813 vrid 4095 pol radius-svr-pol-
icy-1
"

43 2014/06/26 03:02:18.38 UTC MINOR: DEBUG #2001 management RADIUS
"RADIUS: Script
  Accounting-Response(5) id 8 len 20 from 192.168.1.1:1813 policy python-policy-1 status
success
```

```
"
44 2014/06/26 03:07:18.71 UTC MINOR: DEBUG #2001 Base Python Output
"Python Output: python-script-1
"

45 2014/06/26 03:07:18.71 UTC MINOR: DEBUG #2001 Base Python Result
"Python Result: python-script-1
RADIUS Attribute: Type 25, SET
     'Class-1'
     'Class-22'
     'Class-333'
"

46 2014/06/26 03:07:18.71 UTC MINOR: DEBUG #2001 management RADIUS
"RADIUS: Transmit
  Accounting-Request(4) 192.168.1.1:1813 id 9 len 302 vrid 4095 pol radius-svr-policy-1
    STATUS TYPE [40] 4 Interim-Update(3)
    NAS IP ADDRESS [4] 4 192.168.1.254
    USER NAME [1] 17 00:20:fc:1e:cd:53
    SESSION ID [44] 63 00:20:fc:1e:cd:53|1/1/9@1/1/9@sla-profile-1_2014/06/26 03
:02:18
    SESSION TIME [46] 4 300
    EVENT TIMESTAMP [55] 4 1403752038
    VSA [26] 12 Alcatel(6527)
      INPUT_INPROF_OCTETS_64 [19] 10 0x00010000000000000000
    VSA [26] 12 Alcatel(6527)
      INPUT_OUTPROF_OCTETS_64 [20] 10 0x00010000000000000000
    VSA [26] 12 Alcatel(6527)
      INPUT_INPROF_PACKETS_64 [23] 10 0x00010000000000000000
    VSA [26] 12 Alcatel(6527)
      INPUT_OUTPROF_PACKETS_64 [24] 10 0x00010000000000000000
    VSA [26] 12 Alcatel(6527)
      OUTPUT_INPROF_OCTETS_64 [21] 10 0x0001000000000000bea
    VSA [26] 12 Alcatel(6527)
      OUTPUT_OUTPROF_OCTETS_64 [22] 10 0x00010000000000000000
    VSA [26] 12 Alcatel(6527)
      OUTPUT_INPROF_PACKETS_64 [25] 10 0x00010000000000000019
    VSA [26] 12 Alcatel(6527)
      OUTPUT_OUTPROF_PACKETS_64 [26] 10 0x00010000000000000000
    CLASS [25] 7 0x436c6173732d31
    CLASS [25] 8 0x436c6173732d3232
    CLASS [25] 9 0x436c6173732d333333
"

47 2014/06/26 03:07:18.71 UTC MINOR: DEBUG #2001 management RADIUS
"RADIUS: Receive
  Accounting-Response(5) id 9 len 20 from 192.168.1.1:1813 vrid 4095 pol radius-svr-pol-
icy-1
"


48 2014/06/26 03:07:18.71 UTC MINOR: DEBUG #2001 management RADIUS
"RADIUS: Script
  Accounting-Response(5) id 9 len 20 from 192.168.1.1:1813 policy python-policy-1 status
success
"
44 2014/06/26 03:07:18.71 UTC MINOR: DEBUG #2001 Base Python Output
"Python Output: python-script-1
"
```

```
45 2014/06/26 03:07:18.71 UTC MINOR: DEBUG #2001 Base Python Result
"Python Result: python-script-1
RADIUS Attribute: Type 25, SET
     'Class-1'
     'Class-22'
     'Class-333'
"

46 2014/06/26 03:07:18.71 UTC MINOR: DEBUG #2001 management RADIUS
"RADIUS: Transmit
  Accounting-Request(4) 192.168.1.1:1813 id 9 len 302 vrid 4095 pol radius-svr-policy-1
    STATUS TYPE [40] 4 Interim-Update(3)
    NAS IP ADDRESS [4] 4 192.168.1.254
    USER NAME [1] 17 00:20:fc:1e:cd:53
    SESSION ID [44] 63 00:20:fc:1e:cd:53|1/1/9@1/1/9@sla-profile-1_2014/06/26 03
:02:18
    SESSION TIME [46] 4 300
    EVENT TIMESTAMP [55] 4 1403752038
    VSA [26] 12 Alcatel(6527)
      INPUT_INPROF_OCTETS_64 [19] 10 0x00010000000000000000
    VSA [26] 12 Alcatel(6527)
      INPUT_OUTPROF_OCTETS_64 [20] 10 0x00010000000000000000
    VSA [26] 12 Alcatel(6527)
      INPUT_INPROF_PACKETS_64 [23] 10 0x00010000000000000000
    VSA [26] 12 Alcatel(6527)
      INPUT_OUTPROF_PACKETS_64 [24] 10 0x00010000000000000000
    VSA [26] 12 Alcatel(6527)
      OUTPUT_INPROF_OCTETS_64 [21] 10 0x0001000000000000bea
    VSA [26] 12 Alcatel(6527)
      OUTPUT_OUTPROF_OCTETS_64 [22] 10 0x00010000000000000000
    VSA [26] 12 Alcatel(6527)
      OUTPUT_INPROF_PACKETS_64 [25] 10 0x00010000000000000019
    VSA [26] 12 Alcatel(6527)
      OUTPUT_OUTPROF_PACKETS_64 [26] 10 0x00010000000000000000
    CLASS [25] 7 0x436c6173732d31
    CLASS [25] 8 0x436c6173732d3232
    CLASS [25] 9 0x436c6173732d333333
"

47 2014/06/26 03:07:18.71 UTC MINOR: DEBUG #2001 management RADIUS
"RADIUS: Receive
  Accounting-Response(5) id 9 len 20 from 192.168.1.1:1813 vrid 4095 pol radius-svr-pol-
icy-1
"

48 2014/06/26 03:07:18.71 UTC MINOR: DEBUG #2001 management RADIUS
"RADIUS: Script
  Accounting-Response(5) id 9 len 20 from 192.168.1.1:1813 policy python-policy-1 status
success
"
```

As the debug output shows, **python-script-1** stores the three class attributes from the access-accept message which then are reflected into the accounting-start and interim-update messages.

The **tools dump python python-policy** <*name*> **cache** command can be used to show the existing cached entries in the specified python-policy:

```
A:BNG-1# tools dump python python-policy "python-policy-1" cache
===============================================================================
Python policy cache "python-policy-1" entries
===============================================================================
Key       : 00:20:fc:1e:cd:53
Value     : (hex) 03 07 08 09 43 6c 61 73 73 2d 31 43 6c 61 73 73 2d 32 32 43 6c 61 73 73
2d 33 33 33
Time Left : 0d 00:12:08
DDP Key   : N/A
===============================================================================
```

Step 6. Release DHCPv4 lease on the PC

- Release DHCPv4 lease on the PC which is sent to BNG-1.
- DHCPv4 release message from the PC will trigger BNG-1 to remove the ESM host on BNG-1.
- BNG-1 will send an accounting-stop packet to the RADIUS server.

The following is the debug output:

```
"Python Output: python-script-1
"

67 2014/06/26 03:26:14.84 UTC MINOR: DEBUG #2001 Base Python Result
"Python Result: python-script-1
RADIUS Attribute: Type 25, SET
     'Class-1'
     'Class-22'
     'Class-333'
"

68 2014/06/26 03:26:14.85 UTC MINOR: DEBUG #2001 management RADIUS
"RADIUS: Transmit
  Accounting-Request(4) 192.168.1.1:1813 id 13 len 308 vrid 4095 pol radius-svr-
policy-1
    STATUS TYPE [40] 4 Stop(2)
    NAS IP ADDRESS [4] 4 192.168.1.254
    USER NAME [1] 17 00:20:fc:1e:cd:53
    SESSION ID [44] 63 00:20:fc:1e:cd:53|1/1/9@1/1/9@sla-profile-1_2014/06/26 03
:02:18
    SESSION TIME [46] 4 1436
    TERMINATE CAUSE [49] 4 User Request(1)
    EVENT TIMESTAMP [55] 4 1403753174
    VSA [26] 12 Alcatel(6527)
      INPUT_INPROF_OCTETS_64 [19] 10 0x00010000000000000000
    VSA [26] 12 Alcatel(6527)
      INPUT_OUTPROF_OCTETS_64 [20] 10 0x00010000000000000000
    VSA [26] 12 Alcatel(6527)
      INPUT_INPROF_PACKETS_64 [23] 10 0x00010000000000000000
    VSA [26] 12 Alcatel(6527)
      INPUT_OUTPROF_PACKETS_64 [24] 10 0x00010000000000000000
    VSA [26] 12 Alcatel(6527)
      OUTPUT_INPROF_OCTETS_64 [21] 10 0x0001000000000001a36
    VSA [26] 12 Alcatel(6527)
      OUTPUT_OUTPROF_OCTETS_64 [22] 10 0x00010000000000000000
    VSA [26] 12 Alcatel(6527)
      OUTPUT_INPROF_PACKETS_64 [25] 10 0x00010000000000000037
    VSA [26] 12 Alcatel(6527)
      OUTPUT_OUTPROF_PACKETS_64 [26] 10 0x00010000000000000000
    CLASS [25] 7 0x436c6173732d31
    CLASS [25] 8 0x436c6173732d3232
    CLASS [25] 9 0x436c6173732d333333
"

69 2014/06/26 03:26:14.85 UTC MINOR: DEBUG #2001 management RADIUS
"RADIUS: Receive
  Accounting-Response(5) id 13 len 20 from 192.168.1.1:1813 vrid 4095 pol radius
```

```
-svr-policy-1
"

70 2014/06/26 03:26:14.85 UTC MINOR: DEBUG #2001 management RADIUS
"RADIUS: Script
  Accounting-Response(5) id 13 len 20 from 192.168.1.1:1813 policy python-policy
-1 status success
"
```

As the debug output shows, **python-script-1** inserts three RADIUS class attributes in the accounting-stop message.

The **tools dump python python-policy** <*name*> **cache** command can be used to verify the cached entry has been removed:

```
A:BNG-1# tools dump python python-policy "python-policy-1" cache
===============================================================================
Python policy cache "python-policy-1" entries
===============================================================================
===============================================================================
```

Step 7. Manually change the lifetime of a cached entry (optional).

A **tools** command can be used to manually change the lifetime of an existing cached entry, with following syntax:

```
tool perform python-policy <name> cache {hex-key <hex-str>|string-key <str>} set-lifetime
<newlifetime>
```

However, manually changing the lifetime might cause issues with the Python script (for example, if reducing the lifetime causes the entry to expire) that needs the cached entry so it should be used with caution.

To demonstrate this recreate the cached entry by initiating a DHCPv4 discover from the PC (see Step 5, Initiate DHCPv4 on the PC. on page 2591), then change the lifetime to 20 minutes.

```
A:BNG-1# tools dump python python-policy "python-policy-1" cache
===============================================================================
Python policy cache "python-policy-1" entries
===============================================================================
Key       : 00:20:fc:1e:cd:53
Value     : (hex) 03 07 08 09 43 6c 61 73 73 2d 31 43 6c 61 73 73 2d 32 32 43 6c 61 73 73
2d 33 33 33
Time Left : 0d 00:09:48
DDP Key   : N/A
===============================================================================
A:BNG-1# tools perform python-policy "python-policy-1" cache string-key
"00:20:fc:1e:cd:53" set-lifetime 1200
A:BNG-1# tools dump python python-policy "python-policy-1" cache
===============================================================================
Python policy cache "python-policy-1" entries
===============================================================================
Key       : 00:20:fc:1e:cd:53
Value     : (hex) 03 07 08 09 43 6c 61 73 73 2d 31 43 6c 61 73 73 2d 32 32 43 6c 61 73 73
2d 33 33 33
Time Left : 0d 00:19:57
DDP Key   : N/A
===============================================================================
```

Step 8. Manually remove a cached entry (optional).

Manually removing an existing cached entry can be done using a clear command with following syntax:

```
clear python python-policy <name> cache {hex-key <hex-str>|string-key <str>}
```

Manually removing a cached entry can result in unexpected results (for example, if a script expects an entry to exist but it has been removed), so this should be used with caution.

The following command sequence demonstrates the effect of the clear command after initiating a DHCPv4 discover from the PC to recreate the cached entry (see Step 5, Initiate DHCPv4 on the PC. on page 2591).

```
A:BNG-1# tools dump python python-policy "python-policy-1" cache
===============================================================================
Python policy cache "python-policy-1" entries
===============================================================================
Key       : 00:20:fc:1e:cd:53
Value     : (hex) 03 07 08 09 43 6c 61 73 73 2d 31 43 6c 61 73 73 2d 32 32 43 6c 61 73 73
2d 33 33 33
Time Left : 0d 00:11:39
DDP Key   : N/A
===============================================================================
A:BNG-1# clear python python-policy "python-policy-1" cache string-key "00:20:fc:1e:cd:53"
A:BNG-1# tools dump python python-policy "python-policy-1" cache

===============================================================================
Python policy cache "python-policy-1" entries
===============================================================================
===============================================================================
```

Configuration and Operational Guidelines

The following is a list of configuration and operational guidelines that a user should follow when using the Python cache:

- SR OS has a limit on the total amount of memory used for the python cache since the python cache can be demanding with respect to memory usage. The maximum memory allocated for the cache system wide is restricted to 256MB. However, it is good practice to configure per python-policy limits using the **entry-size** and **max-entries** commands; by doing this, one python-policy's cache will not impact another python-policy's cache memory usage.
- For applications needing a cache entry for the entire lifetime of an ESM host, lifetime management is essential. If the lifetime is too long then unneeded entries might reside in the system, wasting memory; if the lifetime is too short then entries might expire while they are still needed. One way to address this is by initially setting a relative short lifetime and then using the RADIUS interim-update message as a trigger to reset a new lifetime. This new lifetime should be larger than the interim-update interval. Then the entry should be removed by a script when an accounting-stop message is sent.
- With MCS enabled, each python cached entry will have a corresponding MCS record, resulting in each python cache entry consuming twice amount of memory. For example, 256MB cached entries would consume additional 256MB memory for MCS records.
- Choosing the right key is important, a network designer needs to choose the key meeting the application requirement in terms of their uniqueness. For example: if an application needs to store per-host information then the key for that host must be unique (for example, its MAC address or remote-id). The key must be derived from the trigger packet. For example: if an application needs to store information on DHCP discovery and retrieve it on receiving a RADIUS accounting-request message for the same host, then the script needs to be able derive the same key from both the DHCP discovery as well as from the RADIUS accounting-request message.
- Using tools or clear commands to manually change cached entries could cause problems if the entries are needed by a script. Only use these commands when it is absolutely necessary.
- The minimum-lifetimes exist to make the system more efficient in handling system memory before a cache entry could be synced or made persistent (e.g. when a new entry is created or when MCS is enabled). The cache entry's lifetime must be equal or larger than the configured minimal-lifetime listed below for that function to occur.

  1. high-availability — The minimum lifetime of a cache entry for it to be synchronized from the active CPM to the standby CPM.
     The default is 0 seconds, resulting in all entries being synchronized.
  2. multi-chassis-redundancy — The minimum lifetime of a cache entry for it to be synchronized between chassis.
     The default is 0 seconds, resulting in all entries being synchronized.

3. persistence — The minimum lifetime of a cache entry for it to be written to the persistency file.
   The default is 0 seconds, resulting in all entries will be synchronized.

Conclusion

The Python cache provides a very powerful and flexible way to share information across different Python scripts in SR OS.