PREV NEXT

ISAKMP/IKE Phase 2 Connections

In this section I'll discuss some router commands you can use to troubleshoot ISAKMP/ IKE Phase 2 connections. I'll begin by describing briefly the commands you can use and then, in later sections, discuss some of these commands in more depth.

Overview of the Phase 2 Commands

If you're experiencing problems with establishing IPsec data connections with an IPsec peer, there are several commands you can use to help pinpoint the problem. Here's a brief summary of these commands:

- **show crypto engine connections active** Displays each data SA that was built and the amount of traffic traversing each.
- **show crypto ipsec sa** Displays the data SAs established between two IPsec peers, and the components used to protect the connections and statistical information.
- **debug crypto isakmp** Displays the steps taken to build a management connection and data connections via the management connection (see "The **debug crypto isakmp** Command" section previously in the chapter).
- **debug crypto engine** Displays events related to encrypting and decrypting packets and applies to both Phase 1 and Phase 2 (see "The **debug crypto engine** Command" section previously in the chapter).
- **debug crypto ipsec** Displays the actual creation of the two unidirectional data SAs between two peers.
- **clear crypto sa** [**counters** | **map** *map_name* | **peer** *IP_address*| **spi** *IP_address* {**ah** | **esp**} *SPI_#*] Clears the statistics (**counters**), all data SAs associated with a crypto map (**map**), all data SAs associated with a peer (**peer**), or a particular data SA to a particular peer.

The following sections will discuss some of these commands in more depth.

The show crypto engine connection active Command

The **show crypto engine connection active** command displays the active SAs (management and data connections) terminated on the router, and the number of data packets encrypted and decrypted for each SA. Example 19-8 illustrates the use of this command.

Example 19-8. Using the *show crypto engine connection active* Command

```
r3640a# show crypto engine connection active
  ID Interface     IP-Address  State Algorithm        Encrypt  Decrypt
   1 Ethernet0/0   192.1.1.40  set   HMAC_SHA+AES_CBC       0        0
2001 Ethernet0/0   192.1.1.40  set   AES+SHA                0        5
2002 Ethernet0/0   192.1.1.40  set   AES+SHA                5        0
```

The first entry (ID #1) is the management connection and the following two entries (ID #2001 and #2002) are the two data connections. A state of "set" indicates that the connections have been fully established.

The show crypto ipsec sa Command

The **show crypto ipsec sa** command displays the crypto map entry information used to build data connections and any existing data connections to remote peers. Example 19-9 illustrates the use of this command. At the top of the display, you can see that the crypto map called "mymap" has been activated on ethernet0/0. The *local ident* and *remote ident* entries display the traffic that is to be protected (traffic between 192.168.2.0/24 and 192.168.3.0/24). The *#pkts encaps* and *#pkts decaps* displays the number of packets encapsulated or de-encapsulated using IPsec (AH or ESP); likewise, you can see the number of packets encrypted and decrypted, and the number of packets where a hash function was created or verified. Given that there are nonzero numbers in these fields, a connection is currently established to the remote peer (192.1.1.42).

Example 19-9. Using the *show crypto ipsec sa* Command

```
r3640a# show crypto ipsec sa
interface: Ethernet0/0
    Crypto map tag: mymap, local addr 192.1.1.40
   protected vrf: (none)
```

```
 local ident (addr/mask/prot/port):  (192.168.2.0/255.255.255.0/0/0)
 remote ident (addr/mask/prot/port): (192.168.3.0/255.255.255.0/0/0)
 current_peer 192.1.1.42 port 500
   PERMIT, flags={origin_is_acl,}
  #pkts encaps: 5, #pkts encrypt: 5, #pkts digest: 5
  #pkts decaps: 5, #pkts decrypt: 5, #pkts verify: 5
  #pkts compressed: 0, #pkts decompressed: 0
  #pkts not compressed: 0, #pkts compr. failed: 0
  #pkts not decompressed: 0, #pkts decompress failed: 0
  #send errors 0, #recv errors 0
   local crypto endpt.: 192.1.1.40, remote crypto endpt.: 192.1.1.42
   path mtu 1500, ip mtu 1500
   current outbound spi: 0x79B5B3BD(2041951165)

   inbound esp sas:
    spi: 0x4D1107A7(1292961703)
      transform: esp-aes esp-sha-hmac ,
      in use settings ={Tunnel, }
      conn id: 2001, flow_id: SW:1, crypto map: mymap
      sa timing: remaining key lifetime (k/sec): (4456557/2479)
      IV size: 16 bytes
      replay detection support: Y
      Status: ACTIVE
   inbound ah sas:
   inbound pcp sas:
   outbound esp sas:
    spi: 0x79B5B3BD(2041951165)
      transform: esp-aes esp-sha-hmac ,
      in use settings ={Tunnel, }
      conn id: 2002, flow_id: SW:2, crypto map: mymap
      sa timing: remaining key lifetime (k/sec): (4456557/2472)
      IV size: 16 bytes
      replay detection support: Y
      Status: ACTIVE
   outbound ah sas:
   outbound pcp sas:
```

The SAs are displayed in separate sections. In this example, only ESP is used, so you can see the SPI values, transforms, and other connection particulars in the *inbound esp sas* and *outbound esp sas* sections of the output. If you don't see anything under these sections, then no data connections have been established. Common problems that might cause this situation are:

- Mismatch in transforms
- Mismatch in crypto ACLs
- Mismatch in addresses the two peers will use for IPsec communications

Further troubleshooting can be done with the **debug crypto ipsec** command.

The debug crypto ipsec Command

If you're experiencing problems establishing the two IPsec data connections between peers, the most common IOS command to troubleshoot the problem is **debug crypto ipsec**. Example 19-10 illustrates the use of this command where the two data connections between two peers are established successfully. In this example, the router is accepting an L2L connection request from a remote peer. The referenced numbers to the right are explained below the example.

Example 19-10. Successfully Established IPsec Data SAs

```
IPSEC(key_engine): got a queue event with 1 kei messages
IPSEC(validate_proposal_request): proposal part #1,                (1)
  (key eng. msg.) INBOUND local= 192.1.1.40, remote= 192.1.1.42,
    local_proxy= 192.168.2.0/255.255.255.0/0/0 (type=4),
    remote_proxy= 192.168.3.0/255.255.255.0/0/0 (type=4),
    protocol= ESP, transform= esp-aes esp-sha-hmac (Tunnel),
    lifedur= 0s and 0kb,
    spi= 0x0(0), conn_id= 0, keysize= 128, flags= 0x2
  Crypto mapdb : proxy_match                                       (2)
        src addr      : 192.168.2.0
        dst addr      : 192.168.3.0
        protocol      : 0
        src port      : 0
        dst port      : 0
IPSEC(key_engine): got a queue event with 1 kei messages
IPSEC(spi_response): getting spi 3754627978 for SA                 (3)
        from 192.1.1.40 to 192.1.1.42 for prot 3
IPSEC(key_engine): got a queue event with 2 kei messages
```

```
IPSEC(initialize_sas): ,
  (key eng. msg.) INBOUND local= 192.1.1.40, remote= 192.1.1.42,
    local_proxy= 192.168.2.0/255.255.255.0/0/0 (type=4),
    remote_proxy= 192.168.3.0/255.255.255.0/0/0 (type=4),
    protocol= ESP, transform= esp-aes esp-sha-hmac (Tunnel),
    lifedur= 3600s and 4608000kb,
    spi= 0xDFCB138A(3754627978), conn_id= 0, keysize= 128,
     flags= 0x2
IPSEC(initialize_sas):,                                          (4)
  (key eng. msg.) OUTBOUND local= 192.1.1.40, remote= 192.1.1.42,
    local_proxy= 192.168.2.0/255.255.255.0/0/0 (type=4),
    remote_proxy= 192.168.3.0/255.255.255.0/0/0 (type=4),
    protocol= ESP, transform= esp-aes esp-sha-hmac (Tunnel),
    lifedur= 3600s and 4608000kb,
    spi= 0x3DC7A592(1036494226), conn_id= 0, keysize= 128,
     flags= 0xA
  Crypto mapdb : proxy_match
        src addr     : 192.168.2.0
        dst addr     : 192.168.3.0
        protocol     : 0
        src port     : 0
        dst port     : 0
IPSEC(crypto_ipsec_sa_find_ident_head): reconnecting with the
     same proxies and 192.1.1.42
IPSEC: Flow_switching Allocated flow for sibling 80000003
IPSEC(policy_db_add_ident): src 192.168.2.0, dest 192.168.3.0,
     dest_port 0
IPSEC(create_sa): sa created,                                    (5)
  (sa) sa_dest= 192.1.1.40, sa_proto= 50,
    sa_spi= 0xDFCB138A(3754627978),
    sa_trans= esp-aes esp-sha-hmac , sa_conn_id= 2002
IPSEC(create_sa): sa created,                                    (6)
  (sa) sa_dest= 192.1.1.42, sa_proto= 50,
    sa_spi= 0x3DC7A592(1036494226),
    sa_trans= esp-aes esp-sha-hmac , sa_conn_id= 2001
IPSEC(key_engine): got a queue event with 1 kei messages
IPSEC(key_engine_enable_outbound): rec'd enable notify           (7)
     from ISAKMP
IPSEC(key_engine_enable_outbound): enable SA with spi 1036494226/50
```

Here's a brief explanation of the output from Example 19-10:

1. A transform was sent from the remote peer to the local router to protect the data SA in the inbound direction. If you look back to Example 19-5, reference 13 in the output from the **debug crypto isakmp** command, you can see the negotiation of the transforms being done for the data connection. At this point, the data SA is being built.

2. The traffic to be proxied is verified (the mirrored crypto ACL): traffic between 192.168.2.0 and 192.168.3.0.

3. An SPI is assigned to the inbound data SA and it is initialized.

4. The outbound data SA is being initialized to the remote peer.

5. The inbound data SA is created.

6. The outbound data SA is created.

7. The remote peer is ready to send data on the local router's outbound data SA.

Mismatched Data Transforms

As you can see from the output in Example 19-10, the debug output is fairly straightforward to interpret. Of course, not all data SAs are built successfully. For example, if you don't have a matching transform for the data connections, you'll see the output in Example 19-11 from the **debug crypto isakmp** and the **debug crypto ipsec** commands. The first message is from the latter **debug** command and the last two messages are from the former command. Use the **show crypto ipsec transform-set** command on the two Cisco IOS routers to determine what transforms have already been created.

Example 19-11. Mismatched IPsec data transforms

```
IPsec (validate_proposal): transform proposal
   (port 3, trans 2, hmac_alg 2) not supported
ISAKMP (0:2) : atts not acceptable. Next payload is 0
ISAKMP (0:2) SA not acceptable
```

Mismatched Crypto ACLs

If the crypto ACLs are not mirrored on the two peers, you'll see debug output from the **debug crypto ipsec** and **debug crypto isakmp** commands shown in Example 19-12. The *proxy identities not supported* message indicates that the crypto ACLs (if routers, PIXs, or ASAs) or network lists (if concentrators) do not match (are not mirrored) on the two IPsec peers.

Example 19-12. Mismatched Crypto ACLs: Not Mirrored

```
IPsec(validate_transform_proposal): proxy identities not supported
ISAKMP: IPsec policy invalidated proposal
ISAKMP (0:2): SA not acceptable!
```

Another set of messages you might see appear in Example 19-13. In this example, one side has host-specific ACL entries (192.168.1.1/32 and 192.168.2.1/32) and the other side has network-specific ACL entries (192.168.1.0/24 and 192.168.2.0/24). This misconfiguration is commonly called an *invalid proxy ID*. Based on the IOS version, you also might see *%CRYPTO-4-RECVD_PKT_INV_IDENTITY* for an error message when there is a non-mirrored ACL condition.

Example 19-13. Mismatched Crypto ACLs: Network Versus Host Match

```
IPSEC(validate_proposal_request): proposal part #1, (key eng. msg.)
     dest= 193.1.1.1, src= 192.1.1.1,
     dest_proxy= 192.168.1.1/255.255.255.255/0/0 (type=4),
     src_proxy= 192.168.2.1/255.255.255.255/0/0 (type=4)
```

Incorrect Peer Address

A less common problem I've seen is where the IP address of the peer has been misconfigured on one of the two ends. For example, if a router has two interfaces it can use to reach a remote peer, and has a crypto map applied to both, whichever interface the router uses to connect to the remote peer is the IP address it would use as its local address. However, if the remote peer doesn't define both IP addresses, but only for one of the interfaces, an error will occur, as shown in Example 19-14, when the local peer uses the unconfigured IP address (on the remote peer).

Example 19-14. Invalid Peer Address

```
IPSEC(validate_proposal): invalid local address 192.1.1.2
ISAKMP (0:3): atts not acceptable. Next payload is 0
ISAKMP (0:3): SA not acceptable!
```

To solve this problem, on the router with multiple interfaces, use the **crypto map** *static_map_name* **local-address** *local_interface_name* command to specify which interface address should be used as the local address; most commonly this is a loopback interface. Another reason that the error in Example 19-14 might occur is if you've applied a crypto map to the wrong interface or forgotten to enable the crypto map at all. Therefore, be sure you have applied the crypto map to the correct interface on your router.

Matching on the Incorrect Crypto Map Entry

Another uncommon problem you might experience is if there are overlapping crypto ACLs on a router, where a match is found for a peer for the *wrong* crypto ACL. This can be very difficult to pinpoint. For example, a router might have two crypto ACLs with overlapping entries like that found in Example 19-15. In this example, crypto ACLs 101 and 102 overlap.

Example 19-15. Overlapping Crypto ACL Entries Example

```
RTRA(config)# access-list 101 permit ip 192.168.1.0 0.0.0.255
                        192.168.2.0 0.0.0.255
RTRA(config)# access-list 102 permit ip host 192.168.1.1
                        host 192.168.2.1
RTRA(config)# crypto map mymap 10 ipsec-isakmp
```

```
RTRA(config-crypto-m)# match address 101
RTRA(config-crypto-m)# set peer 192.1.1.1
RTRA(config-crypto-m)# set transform-set trans1
RTRA(config-crypto-m)# exit
RTRA(config)# crypto map mymap 20 ipsec-isakmp
RTRA(config-crypto-m)# match address 102
RTRA(config-crypto-m)# set peer 192.1.1.2
RTRA(config-crypto-m)# set transform-set trans1
RTRA(config-crypto-m)# exit
```

If RTRA has an IPsec tunnel to 192.1.1.2, but not to 192.1.1.1, and 192.1.1.2 forwards a packet from 192.168.2.1 to 192.168.1.1, it will match the crypto ACL in the first entry, thus causing the error shown in Example 19-16. To solve this problem, put the crypto map entry with the more specific crypto ACL entry or entries before the less specific one; in other words, give the 192.1.1.2 peer a lower crypto map entry number than peer 192.1.1.1.

Example 19-16. Overlapping Crypto ACL Error

```
IPSEC(validate_proposal_request): proposal part #1,
  (key eng. msg.) dest= 200.1.1.1, src= 192.1.1.2,
    dest_proxy= 192.168.1.1/255.255.255.255/0/0 (type=1),
    src_proxy= 192.168.2.1/255.255.255.255/0/0 (type=1),
    protocol= ESP, transform= esp-3des esp-md5-hmac ,
    lifedur= 0s and 0kb,
    spi= 0x0(0), conn_id= 0, keysize= 0, flags= 0x4
IPSEC(validate_transform_proposal): peer address 192.1.1.1 not found
```