CHAPTER-5

NAT DETECTION AND TRAVERSAL MODULE

This section describes NAT Traversal and Detection module in DSMIPv6 implementation.

Figure-5.1: NAT Detection/Traversal

5.1. Module Name & Functionality

NAT (Network Address Translation) is the translation of an Internet Protocol address (IP address) used within one network to a different IP address known within another network. One network is designated the inside network and the other is the outside. In DSMIPv6 the mip6d daemon should bypass NAT, when MN is behind NAT' Ted device in IPv4 FL.

Files used:

- mn.c
- ha.c
- xfrm.c
- mn.h
- ha.h
- xfrm.h
- nat.h
- bcache.h
- bul.h
Process Description

NAT detection is done when the initial Binding Update message is sent from the mobile node to the home agent. When located in an IPv4-only foreign link, the mobile node sends the Binding Update message encapsulated in UDP and IPv4, this is handled in xfrm.c file. The mip6d daemon adds xfrm policy/state for UDP encapsulation for BU packet. When the home agent receives the encapsulated Binding Update, it compares the IPv4 address of the source address field in the IPv4 header with the IPv4 address included in the IPv4 care-of address option. If the two addresses match, no NAT device was in the path. Otherwise, a NAT was detected in the path and the NAT detection option is included in the Binding Acknowledgement. The Binding Acknowledgement, and all future packets, is then encapsulated in UDP and IPv4. Note that the home agent also stores the port numbers and associates them with the mobile node's tunnel in order to forward future packets. This is handled in ha.c file. The mip6d daemon adds the xfrm polices/states for UDP encapsulation of BA and IPv6/IPv4 data traffic. Upon receiving the Binding Acknowledgement with the NAT detection option, the mobile node sets the tunnel to the home agent for UDP encapsulation. Hence, all future packets to the home agent are tunneled in UDP and IPv4. If no NAT device was detected in the path between the mobile node and the home agent then IPv4/IPv6 data traffic is not UDP encapsulated. A mobile node will always tunnel the Binding Updates in UDP when located in an IPv4-only network. Essentially, this process allows for perpetual NAT detection. Similarly, the home agent will encapsulate Binding Acknowledgements in a UDP header whenever the Binding Update is encapsulated in UDP. This is handled in mn.c and xfrm .c file. The mip6d daemon adds xfrm polices/states for UDP encapsulation of IPv6/IPv4 data traffic, when NAT was detected between MN and HA.

5.2. Flowchart

NAT Detection and Traversal flow in Mobile Node given in figure 5.2, 5.3. And NAT Detection and Traversal flow in Home Agent is given in figure 5.4, 5.5.
Outbound Packet Flow

1. Movement of MN to FL (movement.c)

2. IPv4 or IPv6? (mn.c)
   - IPv4 / FL
     - Add outbound and inbound xfrm policy/states for UDP Encapsulation of BU packet (xfrm.c)
     - Send BU UDP Encapsulation Flag (xfrm.c)
     - Send UDP Encapsulation BU Packet with COA option in MH header (mh.c)
   - IPv6 / HL
     - Check flags for sig and data traffic for UDP encapsulation (xfrm.c)
     - No
       - Do Nothing
     - Yes
       - Delete outbound and inbound xfrm policy/states for UDP Encapsulation of BU and IPv6/IPv4 data traffic (xfrm.c)
       - Delete the flags for sig and IPv6/IPv4 data traffic (mn.c)

3. End

Figure-5.2: Outbound Packet Flow in Mobile Node.
Inbound Packet Flow

Start

Wait for BA Packet from HA (mn.c)

UDP Encapsulation BA Packet received on MN side

Yes

MN behind NAT? (mn.c)

MN is behind NAT Device

Add outbound and inbound xfrm policies/states for UDP encapsulation of IPv6/IPv4 data Packet (mn.c)

Set flag for IPv6/IPv4 data traffic (mn.c)

Future IPv6/IPv4 data traffic will be UDP Encapsulated

No

MN is not behind NAT Device. So no need to add xfrm policy/state for UDP encapsulation of IPv4/IPv6 data traffic

Data Packets are encapsulated in IPv6 in IPv4 or IPv4 in IPv4 Tunneling

End

Figure- 5.3: Inbound Packet in Mobile Node.
BU Receive Packet Flow

Start

Add xfrm policy/state to accept incoming BU Packet (xfrm.c)

UDP Encapsulation BU received on HA side

NAT Detection? (ha.c)

✓

Store the Port Number in Order to forward future packets (ha.c)

Add the Inbound/Outbound xfrm policy /state for UDP encapsulation of BA and

Set flags for UDP encapsulation of BA and IPv6/IPv4 data traffic

Add NAT Detection Option in BA Packet

All future traffic will be UDP encapsulated

No need to store Port

Add xfrm policy/state to UDP encapsulation BA Traffic (ha.c)

Set flag for BA signaling (ha.c)

Only BA will be UDP encapsulated and no UDP encapsulation of data traffic

End

Figure-5.4: BU Receive Packet flow in Home Agent.
5.3. Internal Data Structure

Following are the main structures that are being used between some of the important functions in NAT Traversal and Detection.
a) **struct encap_info:**

i) Description:
Structure to store the source IP address and port information, once NAT is detected.

ii) File:
mipv6-daemon-umip-0.4/src/nat.h

iii) Code Snippet:
```c
struct encap_info
{
    struct in_addr src;
    uint16_t port;
};
```

b) **Enum for NAT detection:**

i) Description:
Enumeration used for NAT detection.

ii) File:
mipv6-daemon-umip-0.4/src/nat.h

iii) Code Snippet:
```c
typedef enum
{
    MIP6_NAT_DISABLED,
    MIP6_NAT_ENABLED,
};
```

c) **struct xfrm_selector:**

i) Description:
Xfrm selectors for policy and state used for UDP Encapsulation.

ii) File:
iii) Code Snippet:

```c
struct xfrm_selector
{
    xfrm_address_t daddr;
    xfrm_address_t saddr;
    __be16 dport;
    __be16 dport_mask;
    __be16 sport;
    __be16 sport_mask;
    __u16 family;
    __u8 prefixlen_d;
    __u8 prefixlen_s;
    __u8 proto;
    int if index;
    uid_t user;
};
```

d) **struct xfrm_user_tmpl:**

i) Description:

*Used to create template for xfrm policy for UDP encapsulation.*

ii) File:

```
linux-2.6.28.2/include/linux/xfrm.h
```

iii) Code Snippet:

```c
struct xfrm_user_tmpl
```
e) **struct xfrm_encap_tmpl:**

i) **Description:**

Used to create template for xfrm policy for UDP encapsulation.

ii) **File:**

    linux-2.6.28.2/include/linux/xfrm.h

iii) **Code Snippet:**

```
struct xfrm_encap_tmpl
{
    __u16     encap_type;
    __be16    encap_sport;
    __be16    encap_dport;
    xfrm_address_t encap_oa;
};
```
struct bulentry:

i) Description:

This structure stores information about Binding Update List. The members of this structure are used to set Xfrm policy/states for UDP encapsulation in MN side.

ii) File:

mipv6-daemon-umip-0.4/src/bul.h

iii) Code Snippet:

struct bulentry
{
    struct home_addr_info *home; /* Pointer to home_address structure to which this entry belongs to */
    struct tq_elem tqe; /* Timer queue entry */
    struct in6_addr peer_addr; /* CN / HA address */
    struct in6_addr hoa;
    struct in6_addr coa; /* care-of address of the sent BU */
    int if_coa;
    int if_tunnel; /* Tunnel iface for the BCE */
    int if_tunnel4; /* 4/4 or 6/4 tunnel iface for the BCE */
    int type; /* BUL / NON_MIP_CN / UNREACH */
    uint16_t seq; /* sequence number of the latest BU */
    uint16_t flags; /* BU send flags */
    struct in6_addr last_coa; /* Last good coa */
struct timespec lastsent;
struct timespec lifetime; /* lifetime sent in this BU */
struct timespec delay; /* call back time in ms*/
struct timespec expires; /* Absolute time for timer expire */
struct timespec hard_expire; /* Absolute bulentry expiry time */
int consecutive_resends; /* Number of consecutive BU's resent*/

int8_t coa_changed;
uint8_t wait_ack; /* WAIT / READY */
uint8_t xfrm_state;
uint8_t use_alt_coa; /* whether to use alt. CoA option */
uint8_t dereg; /* for calculating BSA key */
uint8_t do_send_bu; /* send bu / not send bu */
uint8_t behind_nat; /* whether a nat was detected */
uint8_t udp_encap; /* doing UDP encap */

/* Information for return routability */

struct retrout_info rr;
uint8_t Kbm[HMAC_SHA1_KEY_SIZE];

void (*callback)(struct tq_elem *);
void (*ext_cleanup)(struct bulentry *);

};

g) struct bcentry:
i) Description:
This structure stores information about Binding Cache Entry. The members of this structure are used to set Xfrm policy/states for UDP encapsulation in HA side.

ii) File:

mipv6-daemon-umip-0.4/src/bcache.h

iii) Code Snippet:

```
struct bcentry
{
    struct in6_addr our_addr;        /* Address to which we got BU */
    struct in6_addr peer_addr;       /* MN home address IPv6 */
    struct in_addr peer_addr4;       /* MN home address IPv4 */
    struct in6_addr old_coa;         /* Previous care-of address */
    struct in6_addr coa;             /* MN care-of address */
    struct timespec add_time;        /* When was the binding added or modified */
    struct timespec lifetime;        /* lifetime sent in this BU, in seconds */
    struct encap_info nat_info;      /* Information for NAT traversal */
    uint8_t behind_nat;              /* whether a nat was detected */
    uint16_t seqno;                  /* sequence number of the latest BU */
    uint16_t flags;                  /* BU flags */
    uint16_t nonce_coa;
    uint16_t nonce_hoa;
    uint16_t type;                   /* Entry type */
    uint16_t nemo_type;              /* NEMO registration type */
    int unreach;                     /* ICMP dest unreach count */
```
int tunnel; /* 6/6 or 6/4 tunnel interface index */
int tunnel4; /* 4/4 or 4/6 tunnel interface index */
int link; /* Home link interface index */
int id; /* For testing */

/* Following fields are for internal use only */
struct timespec brjastsent; /* BR ratelimit */
int br_count; /* BR ratelimit */
pthread_rwlock_t lock; /* Protects the entry */
struct tq_elem tqe; /* Timer queue entry for expire */
uint8_t xfrmstate; /* MY: status of xfrm state for udp encapsulation in HA */

void (*cleanup)(struct bcentry *bce); /* Clean up bce data */
struct list_head mob_net_prefixes; /* Mobile network prefixes v6*/
struct net_prefix4 *mob_net_prefixes4; /* Mobile network prefixes v4*/

5.4. Internal Methods

Following are the main functions used in mIPv6d for implementation of NAT Traversal and Detection module.

a) udpencap_encap_out_IPv4_traffic_start
i) Function Description:

Routine to add udp encapsulated xfrm policy/state for IPv4 data traffic, when NAT is detected.

ii) Input parameters:

@local: IPv6 local address

@lpreflen: prefix length of local address
@dest: IPv6 peer address
@dpreflen: prefix length of peer address
@proto: Protocol
@type: MH header type
@src: IPv4 local address
@dst: IPv4 peer address
@dir: direction
@spi: Security parameter index

iii) Return Type:
In case of error return integer value less than 0.

iv) Caller:
   ha_udpencap_encap_traffic_start
   mn_recv_ba

v) Defined in File:
   xfrm.c

b) udpencap_encap_out_IPv4_traffic_end

i) Function Description:
   Routine to delete udp encapsulated xfrm policy/state for IPv4 data traffic,
   when NAT is detected.

ii) Input parameters:
   @local: IPv6 local address
   @lpreflen: prefix length of local address
   @dest: IPv6 peer address
   @dpreflen: prefix length of peer address
@proto: Protocol
@type: MH header type
@src: IPv4 local address
@dst: IPv4 peer address
@dir: direction
@spi: Security parameter index

iii) Return Type:

In case of error return integer value less than 0.

iv) Caller:

ha_udpen encaps traffic_end

xfrm_del_bule_dsmip

v) Defined in File:

xfrm.c

c) udpen encaps out traffic_start

i) Function Description:

Routine to install a state and policy to encapsulate some kind of traffic into IPv4/UDP.

ii) Input parameters:

@local: local IPv6 IP
@lpreflen: length of local IP
@dest: destination IPv6 IP
@dpreflen: length of destination IP
@proto: protocol
@type: MH header type     /* Outer ip and udp */
@src: Source IP

@sport: Source IP

@dst: destination IP

@dport: destination port /* Policy */

@prio: priority

@dir: direction

@spi: Security parameter index

iii) Return Type:

In case of error return integer value less than 0.

iv) Caller:

ha_udpencap_encap_traffic_start

mn_recv Ba

v) Defined in File:

xfrm.c

d) udpencap_encap_out_traffic_end

i) Function Description:

Routine to remove state and policy installed in previous function.

ii) Input parameters:

@local: local IPv6 IP

@lpreflen: length of local IP

@dest: destination IPv6 IP

@dprelen: length of destination IP

@protocol: Protocol

@type: MH header type /* Outer ip and udp */
iii) Return Type:

In case of error return integer value less than 0.

iv) Caller:

ha_udpencap_encap_traffic_end
xfrm_del_bule_dsmip

v) Defined in File:

xfrm.c
e) xfrm_state_encap_add

i) Function Description:

Routine to add the XFRM states for IPv4/UDP encapsulation.

ii) Input parameters:

@sel: IPv6 selectors
@proto: protocol
@tmpl: template
@update: add new SA or update old one
@flags: flags set
@v4: IPv4 selectors
@spi: Security parameter index

iii) Return Type:

In case of error return integer value less than 0.
iv) Caller:

xfrm_cn_init

xfrm_pre_bu_add_bule

udppcap_encap_out_traffic_start

udppcap_receive_traffic_start

udppcap_encap_out_IPv4_traffic_start

v) Defined in File:

xfrm.c

f) \texttt{xfrm\_state\_encap\_del}

i) Function Description:

Routine for deletion of udp encapsulation xfrm state.

ii) Input parameters:

@proto: Protocol

@sel: IPv4 selectors

@spi: Security parameter index

iii) Return Type:

In case of error return integer value less than 0.

iv) Caller:

xfrm_en_cleanup

xfrm_udp_encap_delete

udppcap_receive_traffic_end

udppcap_encap_out_traffic_end

udppcap_encap_out_IPv4_traffic_end

v) Defined in File:
xfrm.c

g) xfrm_del_bule_dsmip

i) Function Description:

Routine to delete states and policies related to UDP encapsulation for the BULE.

ii) Input parameters:

@bule: BUL structure

iii) Return Type:

In case of error return integer value less than 0.

iv) Caller:

xfrm_del_bule

v) Defined in File:

xfrm.c

h) xfrm_udp_encap_delete

i) Function Description:

Routine to delete xfrm state for UDP encapsulation, when CoA is freed.

ii) Input parameters:

@CoA: Care of Address

@HoA: Home Address

iii) Return Type:

No return type.

iv) Caller:

md_free_coa

v) Defined in File:

xfrm.c
i) **xfrm_pre_bu_add_bule**

i) Function Description:

This routine is called before sending BU; MN should insert UDP encapsulation policy/state only for BU/BA.

ii) Input parameters:

@bule: BUL structure

iii) Return Type:

In case of error return integer value less than 0.

iv) Caller:

pre_bu_bul_update

v) Defined in File:

xfrm.c

j) **mn_recv_ba**

i) Function Description:

This routine is called, when MN receives BA from HA. From this routine the Xfrm policies/states for UDP encapsulation are added, when NAT is detected.

ii) Input parameters:

@mh: mobility header

@len: length of mobility header

@in: in6_addr_bundle structure

@iif: interface index

iii) Return Type:

No return type.

iv) Caller:

This routine is handler of type mh_handler. It is called when BA is received on MN side by HA.

v) Defined in File:

mn.c
k) **ha_udpencap_encap_traffic_end**

i) Function Description:

Routine to delete policy/state for udp encapsulation of BA, IPv6 and IPv4 data packets. When BU is UDP Encapsulated, BA is also udp encapsulated. If NAT is detected, all future IPv6/IPv4 data traffic is also udp encapsulated.

ii) Input parameters:

@bce: binding cache

iii) Return Type:

In case of error return integer value less than 0.

iv) Caller:

home cleanup

v) Defined in File:

ha.c

l) **ha_udpencap_encap_traffic_start**

i) Function Description:

Routine to add policy/state for udp encapsulation of BA, IPv6 and IPv4 data packets.

When BU is UDP Encapsulated, BA is also udp encapsulated.

If NAT is detected, all future IPv6/IPv4 data traffic is also udp encapsulated.

ii) Input parameters:

@bce: binding cache

iii) Return Type:

In case of error return integer value less than 0.

iv) Caller:

ha_recv_bu_worker
v) Defined in File:
   ha.c

5.5. Event Triggering the Process

When MN moves in IPv4 FL, mip6d code adds xfrm policy/state for UDP encapsulation of BU and sends BU to HA. This processing is handled by routine xfrm_pre_bu_add_bule. When BU is received on HA side, it triggers the NAT detection called. And if NAT is detected, it pushes xfrm policies/states for UDP encapsulation of IPv6/IPv4 data traffic and BA in the kernel; otherwise if NAT is not detected it only pushes policy/state for UDP encapsulation of BA. This processing is handled by routine ha_recv_bu_worker, which further calls routine udpencap_encap_out_traffic_start for adding UDP encapsulation of IPv6 data traffic and routine udpencap_encap_out_IPv4_traffic_start for UDP encapsulation of IPv4 data traffic. When UDP encapsulated BA is received on MN side. The mip6d checks the presence of NAT between MN and HA. If NAT is detected, it pushes xfrm policies/states in kernel for UDP encapsulation of future IPv4/IPv6 data traffic. This processing is handled by routine mn_recv_ba, which further call routine udpencap_encap_out_traffic_start for adding UDP encapsulation of IPv6 data traffic and routine udpencap_encap_out_IPv4_traffic_start for UDP encapsulation of IPv4 data traffic. On MN side to flush the xfrm policies and states for UDP encapsulation, the routine called is xfrm_del_bule_dsmip, which further calls routine udpencap_encap_out_traffic_end to flush policies/states for BU and IPv6 data traffic, and routine udpencap_encap_out_IPv4_traffic_end to flush xfrm policies and states for UDP encapsulated of IPv4 data traffic. On HA side to flush the xfrm policies and states for UDP encapsulation, the routine called is ha_udpencap_encap_traffic_end, which further calls routine udpencap_encap_out_traffic_end to flush policies/states for BU and IPv6 data traffic, and routine udpencap_encap_out_IPv4_traffic_end to flush xfrm policies and states for UDP encapsulated of IPv4 data traffic.