XFRM Interface Experiences, Dislikes and Suggested Changes

J Hadi Salim <hadi@znyx.com>
Motivation

• Frustration with the structuring came as a result of looking at:
  – MIBS, Forces LFB semantics
    • Other Vendors who map nicely to the above but not Linux

• Confusion ...
  – A few things may just have been adapted as is from Kame

• A lot of vomiting ...
New/updates responses/events for Security Associations

Main structure for user->kernel config and kernel->user response and event reports is struct \texttt{xfrm usersa info}

\begin{verbatim}
struct xfrm_usersa_info {
    struct xfrm_selector sel; //relevant on config/report
    struct xfrm_id id; // key, must always be present
    xfrm_address_t saddr; //relevance in config/report
    struct xfrm_lifetime_cfg lft; // relevance on config only
    struct xfrm_lifetime_cur curlft; // relevant only on report
    struct xfrm_stats stats; // relevant only on report
    __u32 seq; // config/response, valid in some cases (acquire/allocspi)
    __u32 reqid; //config/response, valid in some cases
    __u16 family; // related to IP
    __u8 mode; // relevance on tunnel report/config(move to flags)
    __u8 replay_window; // config/report
    __u8 flags; // so far valid only on tunnel config
}
\end{verbatim}

// optional TLVs follow, only appear on a need basis:
\texttt{XFRM\_ALG\_XXX/XFRM\_ENCAP}
Get/Dump/Delete of SA

Main structure for user->kernel config and kernel->user response and event reports is struct `xfrm_usersa_id`

```
struct xfrm_usersa_id {
    xfrm_address_t daddr;
    __u32 spi;
    __u16 family;
    __u8 proto;
};
```

- Looks very similar to the key, `xfrm_id`, used in NEW/UPDATE
- combine and come up with the same header for adding/deleting/getting etc
- Nice simple shim header in the classical tradition of other netlink services
- Fix the optionality issue by having TLVs for other elements
  - Much more extensible

- Clearly breaks ABI!
  - Known apps using the API are iproute2 and Pluto
Other xfrm messages

- Expire, Acquire and ALLOCSPI
  - All describe the SA using `struct xfrm_usersa_info`
    - Meaning same desire of reusing
- Flush is the only odd one out
  - uses `struct xfrm_usersa_flush` which contains protocol to be flushed
    - We could use the same shim header if we keep proto in shim header and just ignore the rest of the messages
Kill `xfrm_address`

- I hate unions for netlink configuration
  - We need to be as close to a wire formatted packet
    - Future work to make netlink distributed
- Introduce XFRM_SRC and XFRM_DST TLVs
  - no different than the ones used to configure FIB
    - XFRM_SRC implies a src address
    - XFRM_DST implies a dst address
    - Length (L in TLV) implicitly declares V4 or V6
Introduce shim header, \textit{xfrm\_sa\_id}

- Used in add/del/get/dump/expire/allocspi/ and all events
  - Do same for flush and only set proto field
- Sacrifice a few bits for consistency

```c
struct xfrm\_sa\_id {
    __u32 spi;
    __u8 family;
    __u8 proto;
    __u16 flags;
};
```

flags (16 bits too much?) :
```
#define XFRM\_STATE\_NOECN 1
#define XFRM\_STATE\_DECAP\_DSCP 2
#define XFRM\_MODE 4 // set implies transport, 1 implies tunnel
#define XFRM\_EXPIRE\_MODE 8 // set implies hard expire, 1 implies soft
```
Introduce new attribute TLVs

- **XFRM_SELECTOR**
  - will contain XFRM_DST and XFRM_SRC TLVs
- **XFRM_LIFETIME_CFG**
- **XFRM_LIFETIME_CUR**
- **XFRMA_STATS**
- **XFRMA_SEQ**
- **XFRMA_REQID**
- **XFRMA_REPLAY_WINDOW**
- **XFRMA_SPI_RANGE**
New SA Message Convention

A NEW/UPD config message, attributes
- XFRM_SELECTOR
- XFRM_LIFETIME_CFG
- XFRMA_SEQ (optionally)
- XFRMA_REQID (optionally)
- XFRMA_REPLAY_WINDOW
- XFRMA_ALG_XXX
- XFRMA_ENCAP(optionally)

note the missing attributes for NEW/UPD config
- XFRM_LIFETIME_CUR
- XFRMA_STATS
New/updates responses/events for Security Policy

Main structure for user->kernel config and kernel->user response and event reports is struct `xfrm_userpolicy_info`

```c
struct xfrm_userpolicy_info {
    struct xfrm_selector sel;  //key
    struct xfrm_lifetime_cfg lft;  // config/response/event
    struct xfrm_lifetime_cur curlft;  // response/event
    __u32 priority;  // config
    __u32 index; //key
    __u8 dir; //key
    __u8 action; //BLOCK if set otherwise allow
    __u8 flags; //only XFRM_POLICY_LOCALOK defined
    __u8 share;
};
```

// optional TLVs follow, only appear on a need basis (good thing):
XFRMA_TTPL
Get/Dump/Delete of SP

Main structure for user->kernel config and kernel->user response and event reports is struct \textit{xfrm_userpolicy_id}

```c
struct xfrm_userpolicy_id {
    struct xfrm_selector sel;
    __u32 index;
    __u8 dir;
};
```

- Again: Looks very similar to the \textit{keys} used in NEW/UPDATE of policy
- combine and come up with the shim header like SA
- Fix the optionality issue by having TLVs for other elements
  - Much more extensible

- Again: Clearly breaks ABI!
  - Known apps using the API are iproute2 and Pluto
Other xfrm policy messages

- Expire, Flush
  - Similar treatment as in SA
    - They all eventually deal with xfrm_userpolicy_info
Introduce shim header, *xfrm_sp_id*

- Used in all config/response/event messages

```c
struct xfrm_sp_id {
    __u32 index;
    __u32 flags;
};
```

flags (32 bits too much?) :
// reserve two bits for XFRM_POLICY_DIR IN/OUT/FWD
// 1 bit for XFRM_EXPIRE_MODE
// reserve bits for actions: ALLOW/BLOCK/
// reserve bits for share
// reserve bits for proto (used only in flush messages)
Attribute TLVs

- Reuse the same TLVs from SA
  - nice

- Introduce one new TLV XFRMP_PRIORITY
  - It holds the policy priority
New SP Message Convention

A NEW/UPD config message, attributes
- XFRM_SELECTOR (always present?)
- XFRM_LIFETIME_CFG
- XFRMP_PRIORITY
- XFRMA_TMPL optionally

nlmsg header

xfrm_sp_id

TLVs depending on message type
Usability: Introduce owner field

• We already have this in current FIB
  – We issue numbers to routing daemons and apps
  – Displaying routes shows who inserted them

• Do the same for SADB and SPD

• For SADB we also need to distinguish phase1 vs phase2 Security Associations
  – A few vendors already do this (CISCO comes to mind)
Usability: MISC

• Need better Stats
  – Just reuse Gen stats we already have
  – Need to figure out how many times a SA or SP has been hit pkt/byte count etc

• Delete ranges instead of just flush for both SP/SA
  – Can currently be done if Manager issues multiple messages to delete specific entries
  – Will be made a lot easier if we have indices for all tables
MISC Comments

- We should probably prefix all TLVs as XFRM_xxxx
  - XFRMA_xxxx to SA specific

- Other classifier should be usable for policies instead of standard vanilla flavored SPD
  - In particular makes sense for policy checking
    - Netfilter/iptables
    - I would like to do it from ingress/egress tc level
Modeling IPSEC Inbound

Some manager in user space
IKE/SNMP/ForCES/CLI

IPSEC packet
Use SPI, proto, dstIP
to search SADB

packet decrypt
+ authentication etc

clrtxt packet

packet policy check

0
1
2

SPD-IN/FWD

OK

DROP
Modeling IPSEC Outbound

Some manager in user space
IKE/SNMP/ForCES/CLI

packet from local or forwarding path
Use std 5 tuple + optionally ifindex to lookup

SPD-OUT

relationship via tmpl/bundle resolution

packet policy selection

DROP/BYPASS

SADB

IPSEC packet

packet auth and/or encryption
Table Management

- Tables management typically utilizing something unique about the table properties, a *key*
  - There maybe more than one key applicable
    - SPD index vs Selector
    - My employee ID vs phone extension vs LastName
  - In case of multiple keys, some may make more sense than others in given conditions
Table Management

- Indices or some number are preferable by management
  - SNMP OIDs, Forces Ids, easier to map to hardware (CAMs, address locations etc)
  - less bandwidth
    - imagine a few thousand of these things being sent
- Offload extra computation from the forwarding path
  - “Insert this at index 100” is easier than “heres a selector and priority, please figure out where to insert it”
    - The manager has the intelligence to figure out a given selector and priority should fit at index 100 after sorting etc
And the point is ....

- SPD has selector + dir vs index as keys
- Problem is the index is not controlled by the manager
  - Manager cannot define it
    - Instead the kernel does
      - Other issues with it such as the way it is defined etc (offtopic)
      - This means an intelligent control/datapath separation where you use as little resources as possible is hard to do
- Recommendations
  - SPD index to be defined by manager
    - Default kernel gives you the next available one
What about SADDB?

- It too should have indices!
- Makes it easier to map SPD to SADDB than the case of using tmpls.
  - Infact a few vendors and organizations like the NPF are defining it this way
MISC: ACQUIREs

- One approach
  - SPD fails to find SA
  - Set a SPD flag to indicate ACQ_IN_PROGRESS
    - Set sequence number to expect back
      - very basic simple larval state
  - Initiate ACQUIRE
    - Queue packet
    - Queue incoming packets up to MAX packets allowed
      - Drop any incoming packets after
  - Acquire resolved
    - protect packets
Potpourri

- CISCO has flags and events for sequence number overflows
- What about PMTU? Flags/display?events?
- What should happen when Links go down?
- Can we have DPD(RFC3706) support in kernel?
  - Would need events to inform km of peer death etc
- GRE/IPSEC config?
- NAT-T for static config?
High Availability/Redundancy

- Clearly static info on SADB/SPD needs syncing
- Dynamic details as well need syncing
  - Keys
  - Lifetime state (bytes/timeouts)
  - Sequence numbers + window state
  - SPI?
  - All the above would need configs like configuring lifetimes, inserting expires etc