MACsec

Encryption for the wired LAN

Networking Services Team, Red Hat

Sabrina Dubroca
sd@queasysnail.net

Netdev1.1, Seville, 2016
Outline

- Introduction to MACsec (architecture, protocol, related standards)
- Linux kernel implementation
- Future work
1 Introduction

1. Introduction

- Overview
- Modes
- Protocol details
What is MACsec

- IEEE standard (802.1AE-2006) for encryption over Ethernet
- Encrypt and authenticate all traffic in a LAN with GCM-AES-128
Why MACsec

- Security within LANs (layer 2) is pretty bad
  - rogue DHCP/router advertisements
  - ARP/ndisc spoofing

- IPsec is L3, cannot protect ARP/ndisc on untrusted links

- Cloud environment: VXLAN
  - Encrypted VXLAN: encryption on the tunnel endpoints, not in the VM ⇒ Tenant has no control over the keys
  - MACsec over VXLAN: encryption in the VM, doesn’t need to be aware of the underlay network
MACsec concepts, architecture, and definitions

**Secure channel (SC)** unidirectional channel
- from one node to many
- sequence of successive, overlapping secure associations

**Secure association (SA)** within a SC
- every frame transmitted over MACsec belongs to one particular SA
- packet number and key are per-SA

**Security Entity (SecY)** instance of the MACsec implementation within a node

**Uncontrolled port** network interface providing insecure service
- MACsec is built on top of this
Configuration and relation with IEEE 802.1X

- option 1: admin can configure SC/SA/keys manually
- option 2: use 802.1X with MACsec extensions
  - MKA (MACsec Key Agreement protocol)
  - discovery of other MACsec nodes
  - setup of SC/SA
  - key generation and distribution
  - synchronization of packet numbers
Encryption and integrity

mandatory integrity+authenticity, optional encryption

- default crypto algorithm: GCM-AES
  - authenticated encryption with additional data
- the entire MACsec packet is always authenticated
- admin can choose whether to use encryption
  - no encryption, integrity/authenticity only: entire MACsec packet as additional data
  - encryption + integrity/authenticity: ethernet + MACsec header as additional data, original payload is encrypted and authenticated
Strict validation

Three possible validation modes for incoming packets:

**Strict** Non-protected, invalid, or impossible to verify (no matching channel configured) frames are dropped

**Check** These frames are counted as “invalid” and accepted, if possible

**Disabled** Incoming frames are simply accepted, if possible

- Encrypted frames cannot be accepted without a matching channel and key
Replay protection

- each frame has a 32-bit packet number
- on RX, the node may validate the PN against the lowest PN it expects to get
- configurable replay window
  - some amount of reordering is acceptable
# Packet format (unprotected frame)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Dest addr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Src addr</td>
<td></td>
<td></td>
<td>Ethertype</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>User data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Packet format (protected frame)

Dest addr

Src addr

MACsec Ethertype

SecTAG

(User) Ethertype

Protected (user) data

...}

ICV
## Packet format (encrypted frame)

<table>
<thead>
<tr>
<th>Dest addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Src addr</td>
</tr>
<tr>
<td>MACsec Ethertype</td>
</tr>
<tr>
<td>SecTAG</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Encrypted data</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>ICV</td>
</tr>
</tbody>
</table>

---

SecTAG format

<table>
<thead>
<tr>
<th>MACsec EtherType</th>
<th>TCI</th>
<th>AN</th>
<th>0</th>
<th>SL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **TCI**: tag control information
- **AN**: association number (SA identifier, 2 bits)
- **SL**: short length, non-zero for frame lengths under 64B
- **SCI**: secure channel identifier, 64 bits
  - 48 bits “system identifier” (MAC address)
  - 16 bits “port number”
SecTAG format: TCI field

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>V=0</td>
<td>ES</td>
<td>SC</td>
<td>SCB</td>
<td>E</td>
<td>C</td>
<td>AN</td>
<td></td>
</tr>
</tbody>
</table>

**SC** SCI present

**E** Encrypted payload

**C** Changed text
Interaction with other protocols and layers

<table>
<thead>
<tr>
<th>EthHdr</th>
<th>VLANHdr</th>
<th>Data</th>
</tr>
</thead>
</table>

**Figure:** unprotected VLAN frame

<table>
<thead>
<tr>
<th>EthHdr</th>
<th>SecTAG</th>
<th>VLANHdr</th>
<th>Data</th>
<th>ICV</th>
</tr>
</thead>
</table>

**Figure:** MACsec-protected VLAN frame

- VLAN tag is part of the encrypted payload
Packet handling: Transmit

**Figure:** Packet coming from the stack

1. push SecTAG
2. compute and append ICV
3. pass down to the underlying device

**Figure:** Packet passed down to the network
Packet handling: Receive

<table>
<thead>
<tr>
<th>Eth Hdr</th>
<th>SecTAG</th>
<th>Data</th>
<th>ICV</th>
</tr>
</thead>
</table>

**Figure:** Packet coming from the network

1. verify packet/SecTAG format
2. check packet number (replay protection, optional)
   - just drop the packet, no feedback to a potential attacker
   - helps defend against DoS attacks: don’t perform heavy computation on obviously wrong packets
3. decrypt/verify ICV
4. re-check packet number (replay protection after decryption)
5. remove ICV, pop SecTAG

<table>
<thead>
<tr>
<th>Eth Hdr</th>
<th>Data</th>
</tr>
</thead>
</table>

**Figure:** Packet passed up the stack
2 Implementation

- First idea: Transparent mode
- Better idea: Full netdevice
- Implementation details
Transparent mode: description

- configure MACsec directly on the (real) netdevice
- all packets that go through the device are transparently encrypted and decrypted

advantages

- no extra overhead of adding more netdevices
- seemed easier from a configuration point of view
- looked like it would “just work”
- qdisc layer sees the original packet (no SecTAG, not encrypted)
Transparent mode: problems

- needs hooks in the normal packet processing path
  (\_\_netif\_receive\_skb\_core, xmit\_one)
  - pretty much a non-starter
- makes it very hard to reject RX packets that were not encrypted (including DHCP)
  - possible with hacks in various places to check that the packet was actually decrypted (clearly unacceptable)
  - or let the user add filtering rules manually
    - not really “transparent”
**Transparent mode: problems**

- tcpdump becomes messy (both encrypted and unencrypted packets are captured)
- harder to properly handle VLANs
- unsolved question: how to use multiple TX channels
  - setup rules that match the (unencrypted) TX packets
  - then configure the MACsec encryption process to use a specific TX channel for these matched packets
Full netdevice: description

- create a new netdevice for each TX channel on a specific device
  - similar to VLANs or macvlans
  - “parent” device sees only the raw packets
    - ie, the encrypted/protected packets for all its children MACsec devices
    - and all the non-protected traffic (802.1X, maybe also some normal LAN traffic)
  - good match for the uncontrolled/controlled port model in the IEEE standards
- uses rx_handler and ndo_start_xmit
Crypto

- uses the kernel’s crypto API for Authenticated Encryption with Additional Data (AEAD)
- can use HW acceleration (aesni) if available
## Structures

**struct macsec_dev**
Private data for MACsec netdevice

**struct macsec_secy**
- SecY parameters (validation mode, SCI)
- list of RX channels

**struct macsec_rx_sc**
SCI, container for the SAs

**struct macsec_[tr]x_sa**
- MACsec SA representation
- key
- statistics
- packet number

**struct macsec_tx_sc**
MACsec TX channel, container for the SAs
Structures
RX and TX: rx_handler

- also used by bond, macvlan, bridge, etc
- if SCI not present in SecTAG: rebuild from MAC address + default port
- find the RX SC that matches the SCI for the received packet on the receiving net_device
  - net_device → SecY list → per-SecY RXSC list
  - the packet goes up the stack with skb->dev set to the net_device for the SecY associated with the matching RXSC
RX and TX: Replay protection

- check the packet number against RX window before decrypting
- check again after decrypting
- then update RX window
RX and TX: `ndo_start_xmit`

- 1-to-1 between the MACsec net_device and the TX secure channel
- encrypt/protect with the currently active SA (`encoding_sa`)
Configuration

- API split between rtnetlink and genetlink
- rtnetlink with MACsec-specific options to create the net_device and configure SecY attributes
- genetlink to configure TXSA, RXSC, RXSA
  - provides demux between the commands for the 3 kinds of objects
  - cleaner API design than if we had to configure everything over rtnetlink
3 Use cases

- Normal use case: LAN
- Normal use case (2): LAN with multiple channels
- Extension: VLAN
- Link aggregation
- In the cloud: VXLAN
MACsec LAN setup

- configure MACsec on the hosts and on each switch port
  - need a switch with MACsec support
- configure MACsec only on the hosts
  - works with any switch
  - switch sees only MACsec-protected traffic
MACsec LAN sample configuration

H1

ip link add link eth0 macsec0 type macsec
ip macsec add macsec0 tx sa 0 on pn 100 key 0 $KEY_0
ip macsec add macsec0 rx address $H2_ADDR port 1

H2

ip link add link eth0 macsec0 type macsec
ip macsec add macsec0 tx sa 0 on pn 100 key 1 $KEY_1
ip macsec add macsec0 rx address $H2_ADDR port 1

Important configuration parameters

Changing the current active TXSA

```plaintext
ip link set macsec0 type macsec encoding 2
```

Enabling encryption (optional)

```plaintext
ip link add link eth0 macsec0 type macsec ...
# setup SA and RX ...

ip link set macsec0 type macsec encrypt on
```

Enabling replay protection (optional)

```plaintext
ip link add link eth0 macsec0 type macsec ...
# setup SA and RX ...

ip link set macsec0 type macsec replay on window 128
```
MACsec LAN setup for multiple secure channels

**Figure:** Example LAN setup with multiple channels

- Nodes H1 and H2 have only one secure channel
  - like in the previous example
- Node H4 has two secure channels
  - different crypto parameters and separate keys for each

Use cases

Multiple channels
Multiple channels on an interface

H4

# channel to H1
ip link add link eth0 macsec0 type macsec
ip macsec add macsec0 tx sa 0 on pn 100 key 1 $KEY_1
ip macsec add macsec0 rx address $H1_ADDR port 1
ip macsec add macsec0 rx address $H1_ADDR port 1 \sa 0 pn 100 on key 0 $KEY_0

# channel to H2
ip link add link eth0 macsec1 type macsec port 2
ip macsec add macsec1 tx sa 0 on pn 400 key 2 $KEY_2
ip macsec add macsec1 rx address $H2_ADDR port 1
ip macsec add macsec1 rx address $H2_ADDR port 1 \sa 0 pn 100 on key 3 $KEY_3
MACsec VLAN setup

**Figure:** Example VLAN setup
VLAN over MACsec configuration (VLAN1)

**H1, VLAN1**

```bash
ip link add link eth0 macsec0 type macsec
ip macsec add macsec0 tx sa 0 on pn 100 key 0 $KEY_0
ip macsec add macsec0 rx address $H2_ADDR port 1
ip macsec add macsec0 rx address $H2_ADDR port 1 \ 
  sa 0 pn 100 on key 1 $KEY_1

ip link add link macsec0 vlan0 type vlan id 42
```

**H2, VLAN1**

```bash
ip link add link eth0 macsec0 type macsec
ip macsec add macsec0 tx sa 0 on pn 100 key 1 $KEY_1
ip macsec add macsec0 rx address $H1_ADDR port 1
ip macsec add macsec0 rx address $H1_ADDR port 1 \ 
  sa 0 pn 100 on key 0 $KEY_0

ip link add link macsec0 vlan0 type vlan id 42
```
VLAN over MACsec configuration (VLAN2)

**H1, VLAN2**

```bash
ip link add link eth0 macsec1 type macsec port 2
ip macsec add macsec1 tx sa 0 on pn 100 key 2 $KEY_2
ip macsec add macsec1 rx address $H2_ADDR port 2
ip macsec add macsec1 rx address $H2_ADDR port 2 \
  sa 0 pn 100 on key 3 $KEY_3

ip link add link macsec1 vlan0 type vlan id 10
```

**H2, VLAN2**

```bash
ip link add link eth0 macsec1 type macsec port 2
ip macsec add macsec1 tx sa 0 on pn 100 key 3 $KEY_3
ip macsec add macsec1 rx address $H1_ADDR port 2
ip macsec add macsec1 rx address $H1_ADDR port 2 \
  sa 0 pn 100 on key 2 $KEY_2

ip link add link macsec1 vlan0 type vlan id 10
```
MACsec Bonding setup

- MACsec is configured separately on each underlying link
- MACsec netdevices are enslaved instead of the real links
- LACP/etc traffic is protected by MACsec

**Figure:** Example Bonding setup
MACsec bond configuration

Create bond

```
# modprobe bonding max_bonds=0
ip link add bond0 type bond [...] 
ip link set bond0 up
```

Set up MACsec on each bonded link

```
ip link add link eth0 macsec0 type macsec ... 
# setup SA and RX on macsec0 like before
ip link add link eth1 macsec1 type macsec ... 
# setup SA and RX on macsec1 like before
```

Add the MACsec devices to the bond

```
ip link set macsec0 master bond0
ip link set macsec1 master bond0
```
MACsec VXLAN setup

Figure: Example VXLAN setup

| ETH | IP | UDP | VXLAN | ETH | SecTAG | Payload... | ICV |

Figure: Encapsulation for a MACsec over VXLAN packet
MACsec VXLAN configuration

VXLAN

```bash
ip link add link vxlan0 type vxlan \
  id 10 group 239.0.0.10 ttl 5 dev eth0

ip link add link vxlan0 macsec0 type macsec ...
# setup SA and RX on macsec0 like before
```
4 Conclusion

- Future work
- End
In the kernel

- optional features
  - confidentiality offset: the first 30 bytes of the packet are only integrity protected
  - additional ciphersuite: GCM-AES-256
- hardware offload (at least for some Intel ixgbe NICs)
- performance improvements
In userspace

- NetworkManager support
- \texttt{wpa_supplicant} already has MKA support, need to hook up the netlink API
  - MKA support: commits 7baec808efb5, 887d9d01abc7, dd10abccc86d
More information

- **IEEE 802.1AE-2006**

- **IEEE 802.1X-2010**

- **Kernel submission (RFCv2 on netdev)**
  http://www.spinics.net/lists/netdev/msg362389.html