USAGI IPv6 IPsec Development for Linux

Mitsuru Kanda
Toshiba Corporation
Communication Platform Laboratory
Corporate Research & Development Center
mk@isl.rdc.toshiba.co.jp

Kazunori Miyazawa
Yokogawa Electric Corporation
Corporate Research and Development Headquarters
Advanced Solitions Research Center
Kazunori.Miyzawa@yokogawa.com

Hiroshi Esaki
The University of Tokyo
Graduate School of Information Science and Technology
hiroshi@wide.ad.jp

Abstract

USAGI project[4] was founded to improve and develop Linux IPv6 stack. We also developed (IPv6) IPsec stack for Linux kernel 2.4 and 2.6 series. we present our (IPv6) IPsec implementation (PE_KEY, IPsec Security Association, Security Policy, output processing, and input processing) for both 2.4 and 2.6.

1 Introduction

IPsec is an archtectural system to provide security services in IP layer. On IPv4 IPsec, it is difficult to introduce because we have to retorofit the existing TCP/IP(v4) stack. But on IPv6 IPsec, it is embedded as a standard feature in the IPv6 specification, which means it's easy to deploy with IPv6. Linux kernel has been included the IPv6 basic feature set since version 2.2. Unfortunately IPv6 IPsec was lacking in it. But the IPv4 IPsec stack is provided by FreeS/WAN project[2] as separated from the mainline kernel (but a lot users who need IPv4 IPsec are using it).

At the beginning of the USAGI project, we discussed the design of the IPsec stack and decided to develop a Linux kernel 2.4 IPsec stack both IPv6 and IPv4 from scratch.

Since Linux kernel 2.5 series, new networking architecture was introduced, called XFRM and/or Stackable Destination. The latest Linux kernel version 2.6 supports IPsec both IP versions by using XFRM.

In this paper, we'd like to talk about IPsec implementation for Linux kernel 2.4 and 2.6, then talk about our plan to develop a new key exchange daemon 'Racoon2' in near the future.

2 IPsec

IP Security (IPsec) architecture is described in RFC2401[8]. IPsec consists of an IP packet processing part and a key exchanging part. IPsec processes a packet based on IPsec Security Policy(SP) and IPsec Security Association(SA). SP indicates which packet should be applied with IPsec, bypassed or dropped. When a packet is indicated 'apply IPsec' by SP, IPsec stack process a packet with some parameters SA includes. SA is identified by a set of an IP destination address, an IPsec protocol and a Security Parameter Index(SPI). SA includes its identifier, algorithm, key, anti-replay counter, lifetime and so on. SA itself represents one IPsec protocol. If we want to apply a packet to multiple IPsec protocols(e.g., AH+ESP), we use more than one SA (called SA bundle). SA is stored in IPsec Security Association Database(SAD). SP is stored in IPsec Security Policy Database(SPD).

There are transfer mode and tunnel mode in IPsec architecture. IPsec transfer mode is used for End-to-End communication. IPsec tunnel mode is mainly used for Security Gateway(SWG) to Security Gateway (e.g.,VPN[1]).

PF_KEY[9] interface is used to register/unregister SA parameters with SAD by the system administrator and the key exchange daemon. In addition, some implementations extend it to register/unregister SP with SPD because there is no specification for SPD interface.

IPsec defines two packet formats, AH and ESP. AH is described in RFC2402[6], which provides connectionless integrity and data origin authentication for whole IP packet. ESP is described in RFC2406[7], which provides confidentiality, data origin authentication, connectionless integrity.

[1] Virtual Private Network

To exchange IPsec key automaticaly, a lot of IPsec systems use Internet Key Exchange Protocol(IKE[5]).

3 IPsec implementation for kernel 2.4

In the Linux IPv4 IPsec world, a lot of people use FreeS/WAN project's implementation. It consists of an in-kernel IPsec processing part, Key Exchange daemon 'Pluto' and some utility commands/scripts.

To run Pluto with small changes on our IPsec kernel implementation and reduce impact for user who use FreeS/WAN implementation, we have decided to keep compatibility with FreeS/WAN's IPsec programming interface between kernel and userland. For this, we use the same PF_KEY interface which FreeS/WAN project extended.

In kernel IPsec packet processing part, we developed AH, ESP, SAD and SPD from scratch.

3.1 PF_KEY interface

PF_KEY(v2), which is described in RFC2367, is key management API mainly for IPsec. PF_KEY is used for handling the IPsec Security Association Database. Additionally we have to handle the IPsec Security Policy Database, but there is no standard for the IPsec Security Policy management API. In FreeS/WAN implementation, PF_KEY interface is extended to manage the IPsec Security Policy Database. Our kernel 2.4 IPsec implementation also uses the same PF_KEY interface as FreeS/WAN's one. It is important to be able to run the FreeS/WAN's userland application (e.g., Pluto) with small changes.

3.2 Encryption and Authentication algorithm

We provide HMAC-SHA1 and HMAC-MD5 for authentication, NULL, DES-CBC, 3DES-CBS and AES for encryption. We thought encryption and authentication algorithm is not only used by IPsec and there are many algorithms so that we consider encryption and authentication algorithm and those interface should have good modularity. We adopted cipher modules which provided by CryptoAPI Project[1].

3.3 Security Association and Security Policy

SA and SP themselves don't depend substantially on the IP version. FreeS/WAN project architecture depends on their special virtual network interface for IPsec because it might focus on IPv4 tunnel mode (Their implementation also provides IPv4 transport mode). Their SA, SP, SAD and SPD also depend on their special virtual network interface. We considered and decided it was not suit to IPv6 because the IPv6 stack needed the neighbor discovery and the auto address configuration in its basic specification. If we had implemented IPv6 IPsec stack with their architecture, we had to implement those basic specification in their special virtual network interface. Therefore we implemented our own SAD and SPD in order to handle both IPv4 and IPv6.

To improve the system performance, Each database will be locked by smallest granularity. And in many cases we use the 'read lock'. SA and SP are managed by the reference counter to prevent used SA from removing by accident.

3.4 IPsec Packet Processing

3.4.1 Output

There are various packet output paths from the IP(v4/6) layer to the network driver layer in Linux kernel networking stack (TCP, UDP/ICMP, and NDP[10] for IPv6). The packets which may be applied IPsec will go through these paths. We had to add IPsec functionality for these output paths, e.g, in IPv6 `ip6_xmit()` for TCP, `ip6_build_xmit()` for UDP/ICMP and `ndisc_send_ns()`/`ndisc_send_rs()` for neighbor discovery packets.

Output process is as follows (as shown in Figure1):

1. check IPsec SP
2. lookup the IPsec SA by the IPsec SP
3. apply IPsec processing to the packet
4. output the packet to the network driver layer

To reduce SA searhing time, we link the SP and the found SA after lookup from the first time.

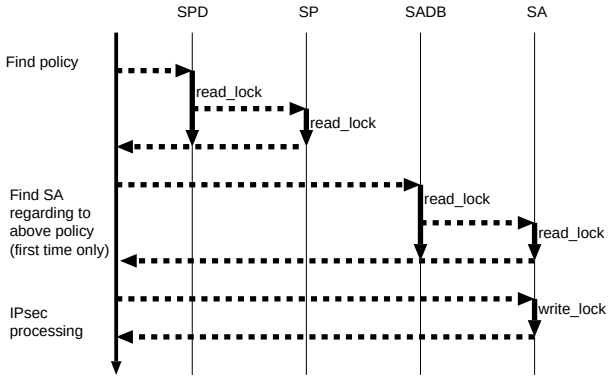

Figure 1. IPsec Output flow

3.4.2 Input

At input, there is only path for IP packets. We added IPsec processing part in `ip6_input_finish()`.

Input process is as follows (as shown in Figure2):

1. receive the packet
2. lookup the IPsec SA by SPI(which resides in AH/ESP header)
3. check integrity and decrypt
4. check IPsec Policy

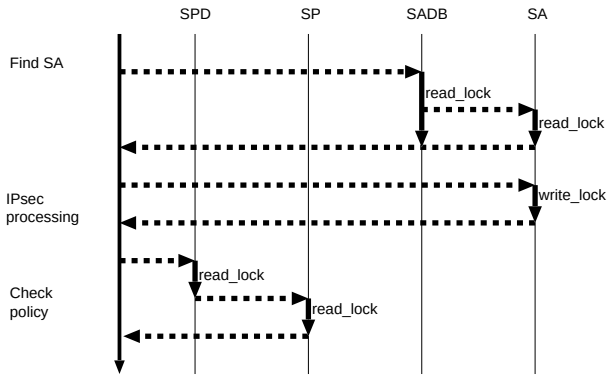

Figure 2. IPsec Input flow

3.5 IPsec Tunnel mode

We are using IPv6-over-IPv6(and IPv4-over-IPv4) virtual tunnel device to implement IPsec tunnel mode. This implementation can avoid to duplication code of encapsulation/decapsulation outer IP header compairing with having these code in the IPsec processing part itself. The virtual tunnel device is not different from the normal IP-over-IP virtual tunnel device in Linux.

4 IPsec implementation for kernel 2.6

In October 2003, we submitted our IPsec code to Linux kernel network maintainers(David S. Miller and Alexey Kuznetsov) for kernel 2.5[2]. At the same time, they were thinking about introducing the new network architecture which was included IPsec support. They liked our implementation for it's simplicity but they thought several details should be handled very much differently. As the result, unfortunately our implementation was not merged into the mainline kernel.

The most important difference between ours and them is SAD/SPD part. They thought the whole SPD/SAD mechanism should be flow cache based lookup system shared by IPv4 and IPv6. One month later, they introduced the new network architecture called 'XFRM' to Linux kernel 2.5. At first their developing code lacked IPv6 IPsec only for IPv4 IPsec. In order to suport IPv6 IPsec, we have implemented IPv6 IPsec code based on XFRM (and discarded our original code).

4.1 PF_KEY interface

The PF_KEY interface of Linux kernel 2.6(and 2.5) is compatible with KAME[3] PF_KEY interface. We can use 'setkey' command for configuring SA and SP and 'Racoon' for IKE. Additionally we can add IPsec Policy each socket via Netlink[3]. They have suported only IPv4 in their first code, we have added IPv6 support.

4.2 Security Association and Security Policy

On the XFRM architecture, IPsec SP, which is represented as `xfrm_policy` structure, will be bound to the routing flow cache (and IPsec policy will point IPsec SA bundle) and IPsec SA, which is represented as `xfrm_state` structure, is included in destination cache, `dst_entry` structure. The chaining destination cache means IPsec SA bundle.

4.3 IPsec Packet Processing

4.3.1 Output

The output part of the XFRM architecture is placed between the IP layer and the network driver layer. In general, non IPsec packet will be passed to the network driver layer by a single destination output function, which is resolved routing lookup. But IPsec packet will be need to apply some IPsec processing (e.g., encryption, hash). XFRM functions make a chain of destination output functions (We call Stackable Destination, as shown in Figure3). Each function match each IPsec processing (AH, ESP and IPcomp[11]).

To be more specific, in order to pass a packet to the network driver layer we have to do as follows (as shown in Figure4):

1. lookup routing table to decide output function by `ip6_route_output()`
2. lookup IPsec Security Policy

[2]kernel version 2.5 series is development version for next stable version 2.6

[3]Netlink is used to transfer network information between kernel and userland applications.

3. lookup IPsec Security Association(s) suitable for IPsec Security Policy and create destination chain

4. to apply IPsec, pass a packet to the destination chain

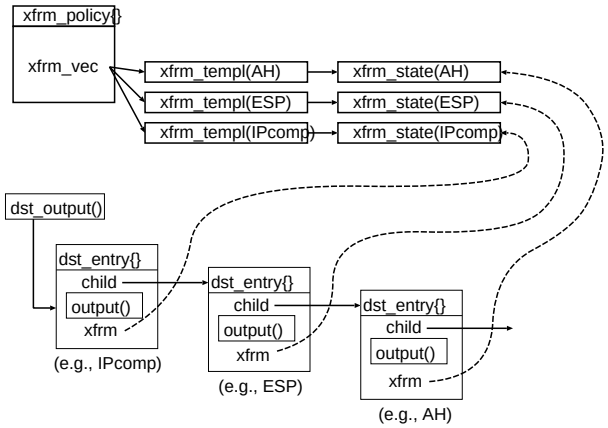

Figure 3. Stackable Destination

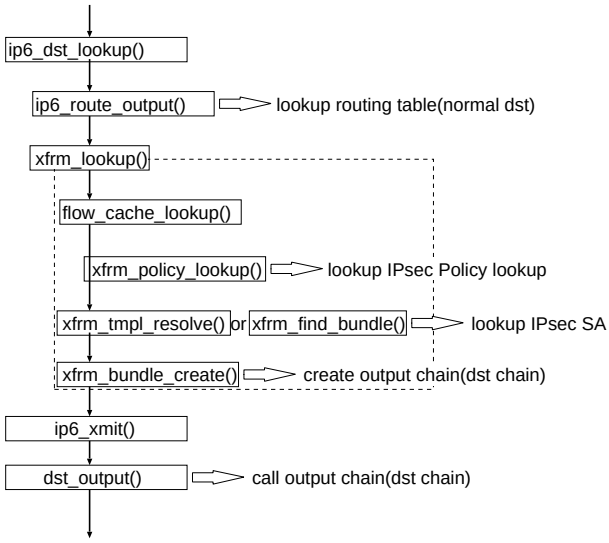

Figure 4. XFRM output flow

4.3.2 Input

The input part of the XFRM architecture is simpler than output. The XFRM input function is handled as same as upper layer protocols like TCP, UDP, etc. In IPv6, IPsec headers are defined as IPv6 extension header but IPsec input functions are handled as an upper layer protocol handler. As the result of introducing IPv6 IPsec input processing in Linux kernel, inconsistencies existed between IPsec headers and other IPv6 extension headers. In order to resolve this, we moved to the other IPv6 extension header handler functions to upper layer protocol handler. In detail, we registered IPsec header (both AH and ESP) handler functions with upper layer protocol handler array `inet6_protos[]`.

Incoming IPsec packet processing flow is as follows (as shown in Figure5):

1. process IP packet from IP header in sequence

2. process IPsec part (check integrity and decrypt) if founded

3. check IPsec Security Policy

4. pass IP packet next handler

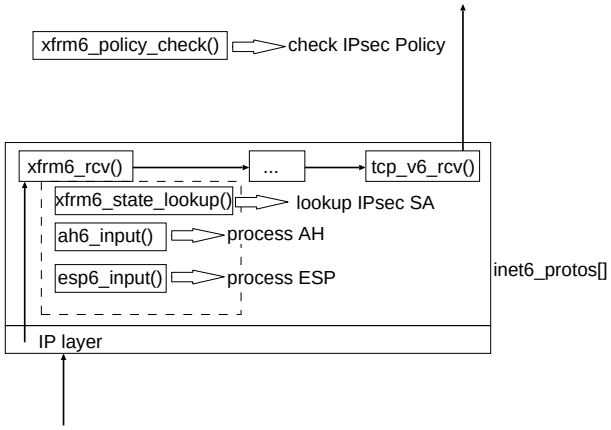

Figure 5. XFRM input flow

4.4 IPsec Tunnel mode

Linux kernel 2.6 IPsec tunnel mode doesn't use the virtual tunnel device to create tunnel. The IPsec stack builds the outer IP header during IPsec processing by itself.

5 Future Work

The XFRM architecture is not only for IPsec but also for other extension headers processing such as Mobile IPv6. We will improve the XFRM architecture to make it be more flexible.

At IETF IP Security working group (IPsec wg), New Internet Key exchange protocol (IKEv2) is now discussed. In order to catch up with the New version of IKE, we have a plan to develop the new key exchange application, Racoon2. IKE is not only Key Exchange protocol for IPsec. Racoon2 will support for multiple Key exchange protocols IKEv1, IKEv2 and Kink.

And also at IETF IPsec wg, new versions of ESP and AH is discussed. we will catch up with them as well.

6 Conclusion

From the new Linux kernel stable version 2.6, we will able to use IPsec by default, there will be no need to add extra patches, recompile kernel.

References

[1] CryptoAPI Project. http://www.kerneli.org/.

[2] FreeS/WAN Project. http://www.freeswan.org/.

[3] KAME Project. http://www.kame.net/.

[4] USAGI Project. http://www.linux-ipv6.org/.

[5] D. C. D. Harkins. The Internet Key Exchange. RFC2409, November 1998.

[6] S. Kent and R. Atkinson. IP Authentication Header. RFC2402, November 1998.

[7] S. Kent and R. Atkinson. IP Encapsulating Security Payload. RFC2406, November 1998.

[8] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. RFC2401, November 1998.

[9] D. McDonald, C. Metz, and B. Phan. PF_KEY Key Management API, Version 2. RFC2367, July 1998.

[10] T. Narten, E. Nordmark, and W. Simpson. Neighbor Discovery for IP Version 6 (IPv6). RFC2461, December 1998.

[11] A. Shacham, B. Monsour, R. Pereira, and M. Thomas. IP Payload Compression Protocol. RFC3173, September 2001.