# Proceedings of the Linux Symposium

## Volume Two

July 19th–22nd, 2006
Ottawa, Ontario
Canada

# Design and Implementation to Support Multiple Key Exchange Protocols for IPsec

Kazunori Miyazawa
*Yokogawa Electric Corporation*
kazunori.miyazawa@jp.yokogawa.com

Shoichi Sakane
*Yokogawa Electric Corporation*
sakane@tanu.org

Ken-ichi Kamada
*Yokogawa Electric Corporation*
ken-ichi.kamada@jp.yokogawa.com

Mitsuru Kanda
*Toshiba Corporation*
mitsuru.kanda@toshiba.co.jp

Atsushi Fukumoto
*Toshiba Corporation*
atsushi.fukumoto@toshiba.co.jp

## Abstract

The racoon2 project has been developing an application, the racoon2, which simultaneously supports multiple key exchange protocols for IPsec [6]. The racoon2 supports IKEv2 [1] and KINK [11], and works on Linux, NetBSD, and FreeBSD. This paper describes issues to support the multiple key exchange protocols on those operating systems, and describes our approach. This paper also describes design and implementation of the racoon2.

## 1 Background

IPsec provides security services in IP layer. To use the services, we need to share IPsec SAs between two entities. IPsec SA consists of a set of security parameters such as IPsec protocol, cipher algorithm, key and so on. There are two methods to share IPsec SAs between two entities. One is manual configuration and the other is automatic key exchange. Manual configuration is basically used for a small static system or debugging because of its scalability. Automatic key exchange is used in a practical system.

We have used the Internet Key Exchange(IKEv1) [2] protocol to support automatic key exchange. But it does not clearly specify the ways to re-key and delete SAs and dead peer detection. The vendors have been extending it to support them and it caused interoperability issues. Additionally, it needs at least 6 messages to exchange IPsec SAs. According to the background IETF IPsec working group had discussed a successor of IKEv1. The working group defines the Internet Key Exchange version 2 (IKEv2) and IETF published it in 2005 as a conclusion of the discussion. IKEv2 reduces the messages to exchange keys from 6 messages to 4 messages. It also specifies to re-key and delete SAs and to detect the dead peer and introduces more functionality.

KINK, Kerberised Internet Negotiation of Keys, is another key exchange protocol. It is defined at KINK working group in IETF. It uses Kerberos to authenticate peers and establishes IPsec SAs only using symmetric key cipher algorithm. Therefore it is available for low-end devices which can not calculate public key algorithm in a practical period. KINK reuses the encoding format of IKEv1 to represent information of IPsec SA so that its payloads are similar to IKEv1.

racoon is widely used as an implementation of IKEv1 on Linux, NetBSD, FreeBSD and others. racoon was developed originally by the KAME project [5] as the implementation on the BSDs. The IPsec-Tools project [4] did porting it on Linux when Linux introduced KAME compatible IPsec stack. The IPsec-Tools project currently maintains and extends it to support various functions.

The racoon2, a successor of racoon, however introduced different architecture and configuration model. The configuration includes IPsec policy to supports the multiple key exchange protocols. Because it tightly links the policy, IPsec SAs, and the key exchange protocol, a user can specify and easily prospect the results of the configuration. It accordingly breaks backward compatibility of the configuration.

In this paper, we discuss issue to support the multiple key exchange protocols in section 2. We describe data structure and architecture of the racoon2 in section 3. We show current status and future works in section 4. We summarize this paper in section 5.

## 2 Supporting the multiple key exchange protocols

We considered two kinds of architecture to implement the multiple key exchange protocols on Linux, NetBSD and FreeBSD operating systems. One is implementing all protocols into single daemon. The other is implementing daemons for each protocol.

We adopted the latter approach. Because single daemon architecture consumes useless resources when user want to use only one protocol. Additionally, it tends to reduce the modularity so that it is possibly difficult to extend to implement new protocols.

The current Linux kernel does not assume to support multiple daemons which process each key exchange protocol. It accordingly can not keep the relationship between an IPsec policy and a key exchange protocol. NetBSD and FreeBSD can not keep the information either. We had had a choice to change the kernels. We however decided to solve the issue within the user-land instead of changing the kernels because of the advantage of deployment of the racoon2.

It is necessary to strictly manage the relationship to get a daemon to process a key exchange request based on a user configuration. Separation of a IPsec policy and the key exchange protocol configuration like racoon does causes possibility of the application to use a different protocol against a key exchange request.

Instead of separated the configuration of racoon, the racoon2 configuration unifies and includes IPsec policies, IPsec SAs and the key exchange protocols. Using this configuration model, user can clearly configure what protocol must be used against the IPsec policy. On the other hand, user can not configure them separately like the usage of racoon.

As mentioned above, the kernel can not keep the relationship because it does not have a field of the key exchange protocol in IPsec policy data structure, which is *struct xfrm_policy* on the Linux. The daemon accordingly needs to search the IPsec policy which triggers the `SADB_ACQUIRE` message, when receiving it.

The PF_KEY [8] API of Linux, NetBSD and FreeBSD contains extension derived from the KAME implementation. The stack has IPsec policy ID. In the Linux kernel the *index* of the *struct xfrm_policy* corresponds it. The *index* is assigned by the kernel and identifies the policy uniquely. The kernel also returns the ID against a request of installing an IPsec policy.

When there is no IPsec SA corresponding the IPsec policy in the kernel, it acquires the IPsec SA by sending a `SADB_ACQUIRE` message to the daemons which listens to PF_KEY socket. KAME extends `SADB_ACQUIRE` message to contain the ID so that the daemon which receives the message can search the IPsec policy which triggers it. As mentioned above, the racoon2 adopts unified configuration model. The daemons can exactly search the original configuration.

## 3 The racoon2

### 3.1 The racoon2 data structure

The data structure basically consists of *selector*, *policy*, *ipsec*, *sa* and *remote*. They are linked by their identifiers. The current racoon2 directly uses this model as its configuration.

- *selector* contains parameters to select traffic through the IPsec stack such as IP addresses, an upper layer protocol, port numbers and so on. *selector* points a *policy* as its action. *selector* is pointed from *remote* when it supports road-warriors. *selector* represents simplex traffic so that there are two *selectors* for an normal bidirectional traffic. The IKE daemon uses the values in *selector* as an IKEv2 Traffic selector payload or an IKEv1 ISAKMP ID payload in phase 2. In KINK protocol, it will be used as a KINK_ISAKMP ID payload.

- *sa* contains information of an IPsec SA. They are an IPsec protocol and candidates of cipher algorithm.

- *ipsec* contains parameter to create IPsec SA bundle. The information consists of common values of bundled IPsec SAs such as lifetime. The racoon2 restricts the type of IPsec SA bundle like the table 3.1. *ipsec* points more than one IPsec SA to create bundle.

| type of bundle | the results packet |
|---|---|
| AH_ESP | [IP][AH][ESP][Payload] |
| AH_IPCOMP | [IP][AH][IPCOMP][Payload] |
| ESP_IPCOMP | [IP][ESP][IPCOMP][Payload] |
| AH_ESP_IPCOMP | [IP][AH][ESP][IPCOMP][Payload] |

Table 1: The types of IPsec SA bundle

- *policy* contains parameter of action against the traffic which matches the selector. The action can be “discard”, “bypass” or “auto_ipsec” to apply IPsec. *policy* also contains mode of IPsec and end point’s addresses if the mode is “tunnel”. a *policy* connects components of the racoon2 data structure. a *policy* points some *ipsec* to make a proposal when the action is “auto_ipsec”.

- *remote* contains parameter for the key exchange protocol. They are identifier of a peer, the IP addresses, the authentication information, algorithm and so on.

A user can flexibly build configuration by linking those components corresponding what user want. For example, in case of that two types of traffic shares a pair of IPsec SA whose proposal is AH and ESP bundle or single ESP, the configuration consists of the components linked like figure 1

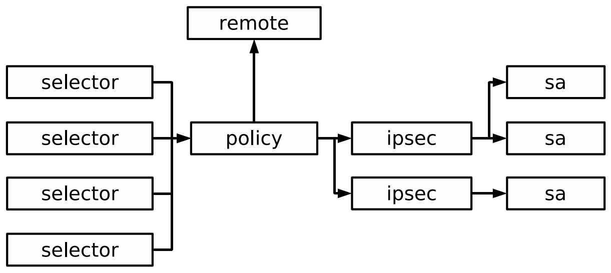

Figure 1: racoon2 data structure

An initiator can retrieve whole configuration from selector. When it is a responder, it can search a *remote* from peer's identifier. If it finds *remote*, it validates the peer and searches the selector from IKEv2 Traffic Selector Payload, ISAKMP ID Payload in IKEv1 phase 2 or KINK_ISAKMP ID Payload. In the case of supporting road-warrior a responder uses link to a *selector* from the *remote* because its address can not be decided in advance.

An responder generates an IPsec policy for the kernel with extracting a chain of *selector*, *policy* and *ipsec*. It generates an IPsec SA as a result of the negotiation with proposals derived from a chain of *policy*, *ipsec* and *sa*.

## 3.2 The racoon2 architecture

The racoon2 consists of 3 daemons. One is spmd, which manages IPsec policy database. Another is iked, which processes IKEv1 and IKEv2 protocol. The other is kinkd, which processes KINK protocol. iked and kinkd are independent from each other. They communicate with spmd via PF_UNIX socket. The kernel broadcasts a `SADB_ACQUIRE` message to all daemons which listens to PF_KEY. iked and kinkd accordingly receive the same `SADB_ACQUIRE` message. The racoon2 adopts the unified configuration model and all daemons read an identical configuration file to share the parameter. The configuration file currently reflects the racoon2 data structure.

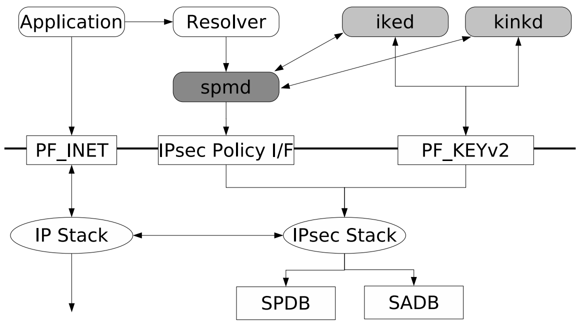

Figure 2: the racoon2 architecture

spmd reads *selector*, *policy* and *ipsec* from the configuration files and installs IPsec policies into the kernel via PF_KEY socket. The kernel returns the message including IPsec policy ID and spmd creates a mapping table of IPsec policy ID and *selector* identifier. Because the daemons on the architecture require the table, spmd must run first.

iked processes IKEv1 and IKEv2 protocol. It should be split to each protocol but iked processes them because those protocols requires same port numbers of UDP.

This is a process sequence when iked is an initiator of IKEv2.

1. The kernel hooks transmission of the traffic which matches the IPsec policy.
2. The kernel sends a `SADB_ACQUIRE` message including IPsec policy ID to the key exchange daemons via PF_KEY socket.
3. iked receives the message and get IPsec policy ID in the `sadb_x_policy_id` field.
4. iked requests the identifier of *selector* corresponding the IPsec policy ID to spmd.

5. iked receives the *selector* identifier from spmd.
6. iked searches *selector* by the identifier and retrieves *policy*, *remote*, *ipsec*, *sa*.
7. iked validates the key exchange protocol in the *remote*.
8. iked processes the acquire according to the protocol in the *remote* such as IKEv2.

The responder processes are listed below when iked is a responder of IKEv2. It depends on whether *remote* includes peers IP address or not.

When remote includes peers IP address, the process like:

1. iked receives a IKE_SA_INIT message.
2. It searches remote by peer's IP address.
3. It replies IKE_SA_INIT using algorithm in the *remote*.
4. It receives IKE_AUTH from the peer.
5. It validates the peer by information in the *remote*
6. It searches the *selector* by the Traffic Selector payload in the message.
7. It finds the *selector* and retrieves *policy*, *ipsec* and *sa*.
8. It processes the request and replies IKE_AUTH

When the remote does not contain the peer's IP address, *e.g.* road-warrior scenario.

1. iked receives a IKE_SA_INIT message.
2. It searches remote by peer's IP address
3. It replies the IKE_SA_INIT message using default algorithm since it can not find specific configuration of the remote.
4. It receives the IKE_AUTH message from the peer.
5. It searches *remote* by the ID payload in the message and authenticates the peer.
6. It also searches *selector* or retrieves the link to *selector* in the *remote*
7. It retrieves linked components, which are *policy*, *ipsec* and *sa*.
8. It processes the request and replies a IKE_AUTH message.

kinkd is a daemon processing KINK protocol. The initiator process of kinkd is similar to iked. kinkd gets an identifier of the *selector* from spmd by indicating a policy_id in the `sadb_x_policy_id` field, and processes the request acquired from the kernel. In the responder process, kinkd always searches the list of *remote* with the identifier of the peer (principal name) because KINK protocol uses Kerberos and kinkd can know the identifier of the peer. When it is a responder, to get the *selector*, kinkd always uses an identifier of *selector* in the *remote* of the configuration. Therefore it currently does not search by KINK_ISAKMP ID payload. After getting the selector, it processes the request from the initiator with linked components.

## 4 Implementation and future works

The racoon2 uses OpenSSL [10] library for its cryptographic operation. It also uses MIT [9] or Heimdal [3] kerberos library to implement kinkd. Current implementation supports IKEv2 and KINK, and does not support IKEv1.

The racoon2 provides a library to support implementing the daemons. The library provides

- configuration file interface
- PF_INET socket utility
- PF_KEYv2 socket utility
- loggin interface
- align differences of OS
- buffer and string utility

So far, the racoon2 provides enough functions to support basic operation on IKEv2, such as

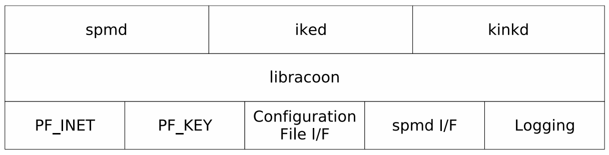

Figure 3: libracoon

- IPsec SA negotiation with IPv4/IPv6 address
- exchange transport mode IPsec SAs with the notification
- dead peer detection
- rekeying
- authentication with either pre-shared-key or certificates
- COOKIE support

Although the racoon2 supports basic functionality on IKEv2, as of this writing, IKEv1 support is still under development. It is important for backward compatibility and it is one of future works. These items are future works of IKEv2:

- improving NAT Traversal
- road-warrior support
- Traffic Selector negotiation
- internal address configuration

Both improving NAT Traversal and supporting road-warrior are especially required for more flexible operation. Both Mobile IPv6 support and MOBIKE are big challenges although they are not listed above. The racoon2 can not work with MIPL-2.0 because MIPL-2.0 maintains IPsec policy by itself. We have a couple of approaches to solve the issue. We, however, need more consideration to decide what is the best strategy.

Concerning KINK protocol, the racoon2 also support enough functions for basic operation:

- IPsec SA negotiation with IPv4/IPv6 address
- optimistic key negotiation
- 3-way key negotiation
- dead peer detection by epoch

There aren't many features left regarding KINK protocol. Kerberos User-to-User authentication mode and KE payload support are a few of them.

Additionally IETF has published new specification of IPsec [7]. The current kernel conforms to the previous specification of IPsec [6] and it will be changed to conform to the new RFC. We probably make the racoon2 coordinate with IPsec stack conforming to the new RFC.

## 5 Summary

We described design and implementation of the racoon2 to support multiple key exchange protocols. And we describe the racoon2 architecture, its data structure and how it works briefly. We also describe the features which have already supported and describe the future works. The racoon2 already have enough functionality on basic key exchange scenario, using IKEv2 and KINK protocols, and we have plan to implement optional functions, including IKEv1 for backward compatibility.

## References

[1] C. Kaufman, Ed. Internet key exchange (ikev2) protocol. RFC4306, December 2005.

[2] D. Harkins and D. Carrel. Internet key exchange (ike) protocol. RFC2409, November 1998.

[3] Heimdal. Heimdal web page. `http://www.pdc.kth.se/heimdal/`.

[4] IPsec Tools. Ipsec tools web page. `http://www.ipsec-tools.sourceforge.net/`.

[5] KAME Project. Kame project web page. `http://www.kame.net`.

[6] S. Kent and R. Atkinson. Security architecture for the internet protocol. RFC2401, November 1998.

[7] S. Kent and K. Seo. Security architecture for the internet protocol. RFC4301, December 2005.

[8] D. McDonald, C. Metz, and B. Phan. Pf_key key management api, version 2. RFC2367, July 1998.

[9] MIT Kerberos. Mit kerberos web page. `http://web.mit.edu/kerberos/www/`.

[10] OpenSSL. Openssl web page. `http://www.openssl.org/`.

[11] S. Sakane, K. Kamada, M. Thomas, and J. Vilhuber. Kerberised internet negotiation of keys (kink). RFC4430, March 2006.