Network Working Group
INTERNET-DRAFT
Category: Informational
<draft-arkko-pfkey-reference-00.txt>
July 14, 2000

Jari Arkko
Ericsson

PF_KEY Extensions for Reducing Policy Information in Kernel

1. Status of this Memo

2. Abstract

PF_KEY is a generic key management API that can be used with IPsec. This document discusses the extension of the PF_KEY interface in order to lessen the need to store complicated IPsec policy data in kernel-mode implementations, and to make it possible for key management daemons reuse traffic pattern information already present in the kernel.

3. Introduction

PF_KEY [1] is a generic key management API that can be used with, for instance, IPsec [2]. PF_KEY is a socket protocol family used by trusted privileged key management applications to communicate with an operating system's kernel-mode implementation of IPsec.

Experience in implementing PF_KEY-based systems has uncovered areas where PF_KEY lacks functionality which is needed for IPsec and IKE [3]. These areas include certain missing algorithms, missing mechanisms to handle IPsec SA bundles, and unnecessary duplication of policy information in several parts of a system. This document describes one possible way to extend PF_KEY. It is targeted as a starting point for discussions and does not claim to solve all known PF_KEY problems.

In this document we discuss the extension of the PF_KEY interface in two respects:

(a) Enabling kernel mode IPsec implementations to know less about IPsec policies, such as what algorithms should be used. At the same time features previously not supported by PF_KEY - such as IPsec SA bundles - can be implemented over PF_KEY in a transparent manner.

(b) Enabling key management daemon implementations to delegate all traffic pattern matching to the kernel. Presently, traffic pattern matching must be performed both in the kernel for outgoing packets and in the key management deamon for incoming IKE connection requests.

4. Overview of Current Operation

Using the existing PF_KEY interface, kernel-mode IPsec implementations can request a key management daemon in user space to create new SAs. The kernel makes such requests using the ACQUIRE message in PF_KEY, and includes the exact list of allowed algorithms and other IPsec parameters. Inside the ACQUIRE message, this list is in a data element called the "Proposal extension". In order to construct the list, the kernel must have a data structure which contains all possible IPsec policy information.

When the peer in an IPsec connection establishes the connection, the process works in a different way. A request from the peer is received by IKE. IKE makes a local decision about what algorithms and parameters are suitable for the proposed traffic type, and once the negotiation is complete, informs the kernel.

As IKE handles only symmetric SAs, the same policy information of (a) what are the algorithms and other parameters and (b) which traffic needs IPsec and with what parameters needs to be stored in two places: the kernel and the key management daemon. This is inconvenient, particularly if there are several key management daemons. For instance, certain Voice over IP architectures require the use of both IKE, Kerberos, and applications as key management daemons. This means that the same information would have to be stored not in two, but several places.

5. Overview of Modified Operation

The current PF_KEY operation is extended in two ways. First, an alternative for the Proposal extension is provided. The purpose of this alternative is to make it possible for the kernel to just reference IPsec policies instead of actually being able to know their contents. This is quite beneficial both because the kernel or hardware IPsec implementation can be made simpler and because PF_KEY no longer needs to suffer about incompatibilities between IKE proposal and PF_KEY proposals. The new alternative extension is called Proposal Reference, and it can appear anywhere where a Proposal extension can in an interchangeable way.

The second extension to PF_KEY operation is to allow the ACQUIRE message also in the direction from the key management daemon to the kernel. Since the kernel must store traffic patterns for detecting e.g. an incoming packet that should have used IPsec but didn't, this information can also be used in other purposes. Namely, upon getting an incoming IKE connection request, the key management daemon must make a decision as to which one of the proposed SAs is suitable, if any. When making this decision it can use the kernel traffic pattern policy information, therefore making it unnecessary to store any of the traffic pattern policy information in the key management deamon. This is useful, for instance, when there can be several key management daemons.

The Proposal Reference extension and the reverse-direction ACQUIRE messsage can also be used together.

6. Messages

6.1. ACQUIRE

The SADB_ACQUIRE message is modified to have the following behaviour:

The kernel sends an SADB_ACQUIRE message to registered sockets.

```
<base, address(SD), (address(P)), (identity(SD),) (sensitivity,)
 proposal-or-propref>
```

The proposal-or-propref must be either the standard PF_KEY Proposal extension, or the Proposal Reference extension defined in this document.

6.2. QUERY

The new SADB_X_QUERY message is sent by the key management deamon in order let the kernel make the decision about a suitable SA. This message resembles ACQUIRE, but is initiated by the key management daemon and, from the point of view of the kernel, is not related to the possibly coming GETSPI, UPDATE, and ADD messages. This message does not create state at the kernel end.

The message behavior of this message is:

Send an SADB_X_QUERY message from a user process to the kernel.

```
<base, address(SD), (address(P), (identity(SD),) (sensitivity,)>
```

The kernel returns the SADB_X_QUERY message to all listening processes.

```
<base, address(SD), (address(P)), (identity(SD),) (sensitivity,)
       proposal-or-propref>
```

The proposal-or-propref must be either the standard PF_KEY Proposal extension, or the Proposal Reference extension defined in this document.

7. Extensions

7.1. Proposal Reference Extension

Like the Proposal extension, the purpose of Proposal Reference extension is to tell the key management daemon the proposal for new SA algorithms and other parameters. It looks like:

```
struct sadb_pref {
        uint16_t sadb_pref_len;
        uint16_t sadb_pref_exttype;
        uint32_t sadb_pref_what;
};
/* sizeof(struct sadb_pref) == 8 */
```

The meaning of the fields is as follows:

sadb_pref_len	This is the length of the extension.
sadb_pref_exttype	This should be SADB_X_EXT_PREFERENCE.
sadb_pref_what	This is an opaque identifier that tells the key management daemon what IPsec policy should be applied. These identifiers have been agreed by the kernel and the key management daemon using mechanisms outside PF_KEY.

Note that the referred policy may request the creation of a simple SA, or even a set of SAs (called an SA bundle). For this reason, key management deamon MUST ignore the sadb_msg_satype field value when interpreting messages containing this extension. The kernel sets sadb_msg_satype to SADB_TYPE_UNSPEC when initiating an ACQUIRE message.

8. Further work

Further discussions are needed in the following areas:

(1) Backwards compatibility. How do the PF_KEY peers know they can use the new extensions?

(2) How should the opaque references be represented in the Proposal

Reference extensions? Is an integer a suitable opaque reference, or would a string be more practical? Perhaps then the kernel traffic pattern configuration could be done completely indepedently of the other configuration; filter definitions could say for instance "if you see 10.x.x.x to 11.x.x.x, apply psec with policy VPN_1".

(3) If multiple key management daemons are assumed, how are proposal references used then? Do the daemons all contact a yet another deamon that holds the mapping from the reference to an actual proposal?

(4) What is the role of future IETF security policy mechanisms in relation to PF_KEY, and do they affect extensions described in this document?

9. Acknowledgements

Possible merit for these extensions should go to the many people with whom I've discussed about these issues, including members of the PF_KEY list.

10. References

[1] D. McDonald, C. Metz, Phan, B. "PF_KEY Key Management API, Version 2" RFC 2367, Sun Microsystems, U.S. Naval Research Laboratory, July 1998.

[2] S. Kent, Atkinson, R. "Security Architecture for the Internet Protocol" RFC 2401, BBN Corp, @Home Network, November 1998.

[3] Harkins, D. and Carrel, D., "The Internet Key Exchange", RFC 2409, Cisco Systems, November 1998.

11. Author's Address

Jari Arkko
Oy LM Ericsson Ab
02420 Jorvas
Finland

Phone: +358 40 5079256 (hand)
+358 9 2992480 (desk)
EMail: Jari.Arkko@ericsson.com
(July 1 - August 30, 2000: jari@arkko.com)