Network Working Group
Internet-Draft
Expires: May 6, 2009

M. Richardson
SSW

November 2, 2008

An abstract interface between applications and IPsec
draft-ietf-btns-abstract-api-02.txt

Status of this Memo

Copyright Notice

Abstract

This document explains in the abstract (no language bindings are provided) how an application may learn that IPsec has been applied to a conversation or specify that IPsec should be used. Though this is useful in general it is particularly useful for applications that wish to use BTNS (Better Than Nothing Security -- a mode of IPsec keying), either in conjunction with channel binding or otherwise.

Table of Contents

1. Overview

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119 [RFC2119].

2. Introduction

Implementation of application protocols that depend on IPsec [RFC4301] tend to depend on configuration of IPsec, without having any portable (or even non-portable) way to ensure that IPsec is being used properly. This state of affairs is unfortunate, as it limits use of IPsec and encourages applications not to rely on IPsec, which in environments that do use IPsec, may lead to redundant cryptographic protection layers.

This document describes an abstract application programming interface (API) that is intended to interface applications with IPsec. It is abstract in that no programming language specific bindings are given here, nor is this API specified in terms of familiar APIs such as the "BSD sockets API," for example. Programming language specific bindings, and operating system specific bindings are left to other documents.

3. Objects involved

There are two major kinds of objects that are defined by this document. These are the Protection Token (pToken) and the Identity Token (iToken). Both objects are abstracted into unique opaque tokens which may be manipulated only indirectly by applications. Here we use the term "opaque token" to mean much what "object" means in a typical object-oriented programming language, but with no public fields (only methods or generic functions). Additionally, the iToken may be serialized -- that is, converted, by application of a suitable function, into an octet string that can later be imported to create a new iToken object that is equivalent to the original (though a value equality test applied to both iTokens may fail).

Each object has a series of attributes associated with it. The API provides a mechanism to query the value of attributes of the token. The attributes are where all of the content of the objects are.

Each token has a scope - the place and time in which it can be considered valid. There are many conflicting qualities that one would wish for the token, and the result is a different compromise among these qualities for each token type. The tokens should be:

- easy to allocate and release
- automatically cleaned up when an application terminates (both properly and improperly)
- easily compared (for equivalence)
- easily interfaced with existing APIs (such as the BSD sockets API, in that case as "auxiliary data")

We use terms such as "process" and "address space" without explaining them or providing references, much as with "object." The terms refer to pervasive, common concepts in operating systems theory and practice over the past several decades.

3.1. Scope of Protection Token

The protection token has a per-process (i.e. per-address space) scope, though it may be inherited by child processes in operating systems that have a "fork()" operation. It SHOULD always be possible to obtain a current protection token for an established connection (whether for a connection-oriented transport protocol or for a "connected" UDP socket). that is equivalent to any previous protection token that was obtained. The scope of the token is not related to any specific underlying Security Associations used by

IPsec, but to the entire set of past, current and future SAs that will be used by IPsec to protect that connection [I-D.ietf-btns-connection-latching].

3.2. Scope of Identity Token

The identity token also has a per-process scope, but is serializable such that its serialized form has a per-system or even universal, scope. (We have to consider whether we want universal scope for serialized iTokens, much as with exported name objects in the GSS-API, which would mean agreeing on a standard, extensible representation and encoding.)

3.3. Validity period of Protection Token

A pToken is valid only within the scope of a single process (though it may be inherited by child processes which share the parent's address space with copy on write semantics). The token may not be serialized, and, therefore, may not be saved in any long term storage.

It is permitted for one protection token to be replaced with another (equivalent) protection token due to a node moving, suspending and resuming, or due to extended network outages, however the underlying identity token would be guaranteed to be the same. This would most likely occur with unconnected sockets, where due to the outage/ downtime, the keying system was unable to maintain a keying channel, and had to re-create the keys from scratch.

3.4. Validity period of Identity Token

The iToken may be valid across the entire system, although it may need to be turned into an external representation (serialization). Some forms of identity token may be valid across systems, but in general an identity token is only valid in reference to a local policy. (See [RFC2692]).

3.5. Serialization

Serialization refers to the process of turning an in memory object into a format which can be saved on disk, and re-imported by the same implementation. This document does not require a specification for the serialization format, only that it be possible. The format is a local matter.

3.5.1. Serialization of Protection Token

There is no requirement to serialize the protection token, or the

attributes contained within. There is a desire to serialize templates for protection tokens such that a set of minimum security requirements can be saved for future connections to the same peer.

3.5.2. Serialization of Identity Token

There is a desire to be able to to serialize the identity token in such a way that future communications can be confirmed to be with the same identity as before.

4. Name space

All symbols (functions, macros, etc.) defined by this API are prefixed with "ipsec_". Specific rules for capitalizations should be driven by the specific language binding.

Whenever sensible, the enumerated values defined in [RFC2367] are used if appropriate.

5. pToken discovery

An application that receives a connection using accept(2) (or recvmsg(2)), or makes a connection using connect(2), needs to get a protection token that is associated with the socket.

For connected sockets (UDP, TCP, some SCTP modes, etc.), the protection token MUST not change during the lifetime of the socket, so a simple process is appropriate.
([I-D.ietf-btns-connection-latching])

As the pToken will not change during the connection. (see notes about re-keying). A simple function is provided to return a pToken from a file descriptor. Many implementations are likely to implement this using getsockopt(2), but an interface in those terms is not specified in order to keep it more abstract, and therefore more portable.

For unconnected sockets (such as UDP and some SCTP modes), each datagram received may be received may arrive from a different source, and therefore may have different protections applied. A protection token needs to be returned with each datagram, so it must be returned as ancillary data with recvmsg(2).

A server using unconnected sockets, would receive a protection token as ancillary data, and then would provide the same protection token as ancillary data on the corresponding sendmsg(2) call.

6. pToken templates

A pToken template is a type of pToken which is used only when setting up a connection, or setting up a socket to listen for connections.

Properties which are not set on a pToken, are assumed to be do-not-care values.

7. Properties of pToken objects

privacyProtected - boolean. Set to false if the connection has either no privacy configured (AH, ESP-null), or if the privacy configured is known to be untrustworthy by the administrator. Returns true otherwise. (XXX: False does not mean that there will be no IPsec, but that it should not be considered useful)

integrityProtected - boolean. Set to false if there is no data integrity protection other than the UDP/TCP checksum.

compressionAvailable - boolean. Set to true if data count sent/ received from socket may not map linearly to data sent/received on wire.

policyName - string. A handle which describes the system policy which was used (or is desired), to establish the connection. This is a string, such as: "secure", "ospf", "iSCSI", "very-secure", "do-not-tell-mom-secure", "minimum-security", "was-posted-on-usenet-security".

iToken - object. Set to iToken object which represents identity of remote system.

remote_iToken - object. Set to iToken object which was used to represent our identity to the remote system.

tunnelMode - boolean. Set if tunnel mode was used, or if it is desired.

ipoptionsProtected - boolean. Set if ip options (and IPv6 header extensions), are protected.

auditString - string. readonly. Not part of a template. Valid only after connection establishment. Contains a string which can be used in system auditing and logging functions which describes the details of the IPsec SA that was negotiated. No structure of this string may be assumed. No session keys are disclosed by this string.

informationString - string. readonly. Not part of a template. Valid only after connection establishment. Contains a string which can be displayed to a user, informing them of what kind of security association was established for this connection. This string may be localized. No session keys are disclosed by this string.

8. Properties of iToken objects

auditString - string. readonly on responder and readonly on initiator after connection establishment. Contains a string which can be used in system auditing and logging functions which describes the remote identity, and the method by which it was authenticated (i.e. it may list the CA or origin of a public key)

authenticationMethod - enumerated type. Indicates which method was used to authenticate the peer, possible values are:

NONE - the peer was not authenticated in anyway

BTNS - the peer was authenticated using an inline key which was not verified in anyway

LEAPOFFAITH - the peer was authenticated using a key which was previously cached, but was previously received inline, and was not verified in anyway.

PRESHAREDKEY - the peer was authenticated using a unique pre-shared key

GROUPKEY - the peer was authenticated using a non-unique pre-shared key

XAUTH - the type of phase1/PARENT-SA is not relevant, as the peer was authenticated using a username/password.

EAP - the type of phase1/PARENT-SA is not relevant, as the peer was authenticated using an EAP method. (Additional properties may provide more information)

PKIX_TRUSTED - the peer was authenticated using a PKIX/X.509 certificate that was found in the trusted store.

PKIX_INLINE - the peer was authenticated using a PKIX/X.509 certificate that was transmitted inline, and was verified by using a Certificate Authority that was found in the trusted store.

PKIX_OFFLINE - the peer was authenticated using a PKIX/X.509 certificate that was retrieved out-of-band (such as by LDAP or HTTP), and was verified by using a Certificate Authority that was found in the trusted store.

certificateAuthorityDN - string. readonly. the Distinguished Name (DN) of certificate authority that was used to verify the key (for methods that involved PKIX)

certificateDN - string. readonly. the DN of the peer that was authenticated

pubKeyID - string. readonly. a somewhat unique identifier for the public key. A suggestion is to use the first 9 base64 digits of the RFC3110 public key modulus, but this is a local matter.

channelBinding - binary blog. readonly. provides the concatenated set of public keys

9. Accessors Functions

Methods to access the properties of the two objects are specific to the language in which the bindings are done. See YYYY for C-bindings.

10. Use Cases

Explain slides from IETF68.

11. Security Considerations

Probably lots to say here. Please help.

12. IANA Considerations

There are no registries created by this document. The names (and language specific enum, if applicable) of the pToken and iToken properties are internal to a single system, and therefore do not need standardization.

13. Acknowledgments

stuff

14. TRACKING

Document RCS tracking info

```
$Revision: 1.7 $
$Log: ietf-btns-abstract-api.xml,v $
Revision 1.7  2008/11/03 02:45:52  mcr
   spell check and updated references

Revision 1.6  2008/02/18 02:37:45  mcr
updated edits.

Revision 1.5  2007/07/24 22:15:51  nico

New abstract, new intro, various minor changes (scope of objects,
etc...).

Revision 1.4  2007/07/24 03:30:19  mcr
        edits to token scope, in collaboration with Nico.

Revision 1.3  2007/07/19 20:09:50  mcr
        added more properties to describe the type of the SA.

Revision 1.2  2007/07/19 19:45:55  mcr
        edits from 2007-07-19 discussion.

Revision 1.1  2007/06/25 15:34:08  mcr
   renamed drafts in Makefile

Revision 1.3  2007/05/14 19:56:37  mcr
  added abstract

Revision 1.2  2007/05/12 20:38:56  mcr
  fixed id string

Revision 1.1  2007/05/12 01:31:00  mcr
  updates to abstract api document

Revision 1.4 2007/02/16 03:24:09 mcr
      updated to make XML happy, and dates corrected
Revision 1.3 2007/02/16 03:04:44 mcr
      C API document.
Revision 1.2 2006/03/21 22:02:47 mcr
      added API requirements and skeleton of original API spec
Revision 1.1 2006/03/21 21:04:43 mcr
      added documents from ipsp WG
Revision 1.1 2003/06/03 20:45:06 mcr
```

initial template

Figure 1: document tracking

15. References

15.1. Normative references

[I-D.ietf-btns-connection-latching]
Williams, N., "IPsec Channels: Connection Latching", draft-ietf-btns-connection-latching-07 (work in progress), April 2008.

[I-D.ietf-btns-core]
Williams, N. and M. Richardson, "Better-Than-Nothing-Security: An Unauthenticated Mode of IPsec", draft-ietf-btns-core-07 (work in progress), August 2008.

[I-D.ietf-btns-prob-and-applic]
Touch, J., Black, D., and Y. Wang, "Problem and Applicability Statement for Better Than Nothing Security (BTNS)", draft-ietf-btns-prob-and-applic-07 (work in progress), July 2008.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2367] McDonald, D., Metz, C., and B. Phan, "PF_KEY Key Management API, Version 2", RFC 2367, July 1998.

[RFC2692] Ellison, C., "SPKI Requirements", RFC 2692, September 1999.

15.2. Non-normative references

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

Author's Address

Michael C. Richardson
Sandelman Software Works
470 Dawson Avenue
Ottawa, ON K1Z 5V7
CA

Email: mcr@sandelman.ottawa.on.ca
URI: http://www.sandelman.ottawa.on.ca/

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).