Network Working Group
Internet-Draft
Expires: December 27, 2007

M. Richardson
SSW

June 25, 2007

An interface between applications and keying systems
draft-ietf-btns-abstract-api-00.txt

Status of this Memo

Copyright Notice

Abstract

The "BTNS" (Better Than Nothing Security) protocols specifies how to use IKEv1 and IKEv2 to do unauthenticated IPsec. This document explains in the abstract (no language bindings are provided) how an application may learn that BTNS IPsec has been applied to a conversation, such that the application can plan to do it's own authentication using a channel binding. In addition, applications can use this API (Application Programming Interface) to request BTNS treatment of the applications' connections.

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

Purpose of this API.

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.

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:

small

easy to allocate and deallocate

automatically cleaned up when an application terminates (both properly and inproperly)

easily compared

easily passed back in a recvmsg(2) call as auxiliary data (for pToken)

3.1. Scope of Protection Token

The protection token has a per-process (i.e. per-address space) scope. The scope of the token is not related to the underlying protection provided by IPsec. The token is a handle.

3.2. Scope of Identity Token

The identity token has a per-system scope, although two applications running on the same system may not be able to compare it literally.

3.3. Validity period of Protection Token

The pToken is valid only within the scope of a single process. The token 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. Some forms of identity token may be valid across systems, but in general an identity token is only valid in reference to a local set of trust anchors. (See [RFC2692]).

4. Namespace

All functions and macros defined by this API are prefixed with "ipsec_" for functions and variables, and with "IPSEC_" if they are macros or enumerated types. (cf. to appropriate POSIX section?)

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 with recvmsg(2) needs to get a protection token that is associated with the socket.

For connected sockets (such as TCP and some SCTP modes), the protection token should not change during the lifetime of the socket, so a simple process is appropriate.

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 ancilliary data with recvmsg(2).

For connected sockets, the pToken will not change during the connection. (see notes about rekeying). A simple function is provided to return a pToken from a file descriptor. Many implementions 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.

6. Properties of pToken objects

privacyProtected - boolean. readonly on responder, get/set on initiator. 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. readonly on responder, get/set on initiator. set to false if there is no data integrity protection other than the UDP/TCP checksum.

compressionAvailable - boolean. readonly on responder, get/set on initiator. Set to true if data count sent/received from socket may not map linearly to data sent/received on wire.

iToken - object. readonly on responder. get/set on initiator. Set to iToken object which represents identity of remote system.

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 details of the IPsec SA that was negotiated. No structure of this string may be assumed. No session keys are disclosed by this string.

7. 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

LEAFOFFAITH - 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

8. Accessor 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.

9. Security Considerations

Probably lots to say here. Please help.

10. IANA Considerations

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

11. Acknowledgments

stuff

12. TRACKING

Document RCS tracking info

```
$Revision: 1.1 $
$Log: ietf-btns-abstract-api-00.txt,v $
Revision 1.1  2007/06/25 15:37:06  mcr
  added ietf-btns-abstract-api

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

13. References

13.1. Normative references

[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.

13.2. 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).