NETWORK WORKING GROUP   N. Williams
Internet-Draft   Sun
Expires: August 27, 2008   February 24, 2008

IPsec Channels: Connection Latching
draft-ietf-btns-connection-latching-06.txt

Status of this Memo

Copyright Notice

Abstract

This document specifies, abstractly, how to interface applications and transport protocols with IPsec so as to create "channels" by "latching" "connections" (packet flows) to certain IPsec Security Association (SA) parameters for the lifetime of the connections. This can be used to protect applications against accidentally exposing live packet flows to unintended peers, whether as the result of a reconfiguration of IPsec or as the result of using weak peer identity to peer address associations.

Weak association of peer ID and peer addresses is at the core of Better Than Nothing Security (BTNS), thus connection latching can add a significant measure of protection to BTNS IPsec nodes. A model of of connection latching is given.

Table of Contents

1.   Introduction 3
1.1. Conventions used in this document 4
2.   Connection Latching 5
2.1. Connection latch state machine 8
2.2. Normative Model: ULP interfaces to the key manager 9
2.3. Informative model: local packet tagging 13
2.4. Non-native mode IPsec 14
2.5. Conflict Resolution 15
3.   Optional protection 16
4.   Simulataneous latch establishment 17
5.   Security Considerations 18
6.   IANA Considerations 19
7.   Acknowledgements 20
8.   References 21
8.1. Normative References 21
8.2. Informative References 21
      Author's Address 23
      Intellectual Property and Copyright Statements 24

1. Introduction

IPsec protects packets with little or no regard for stateful packet flows associated with upper layer protocols (ULPs). This exposes applications that rely on IPsec for session protection to risks associated with changing IPsec configurations, configurations that allow multiple peers access to the same addresses, and/or weak association of peer IDs and their addresses. The latter can occur as a result of "wildcard" matching in the IPsec Peer Authorization Database (PAD), particularly when BTNS [I-D.ietf-btns-prob-and-applic] is used.

Applications that wish to use IPsec may have to ensure that local policy on the various end-points is configured appropriately [I-D.bellovin-useipsec] [I-D.dondeti-useipsec-430x]. There are no standard Application Programming Interfaces (APIs) to do this -- a major consequence of which, for example, is that applications must still use hostnames (and, e.g., the Domain Name System [RFC1034]) and IP addresses in existing APIs and must depend on an IPsec configuration that they may not be able to verify. In addition to specifying aspects of required SPD configuration, application specifications must also address PAD/SPD configuration to strongly bind individual addresses to individual IPsec identities and credentials (certificates, public keys, ...).

IPsec is, then, quite cumbersome for use by applications. To address this we need APIs to IPsec. Not merely APIs for configuring IPsec, but also APIs that are similar to the existing IP APIs (e.g., "BSD sockets"), so that typical applications making use of UDP and TCP can make use of IPsec with minimal changes.

This document describes the foundation for IPsec APIs that UDP and TCP applications can use: a way to bind the traffic flows for, e.g., TCP connections to security properties desired by the application. We call these "connection latches" (and, in some contexts, "IPsec channels"). The methods outlined below achieve this by interfacing ULPs and applications to IPsec.

If widely adopted, connection latching could make application use of IPsec much simpler, at least for certain classes of applications.

Note: the terms "connection latch" and "IPsec channel" are used interchangeably below. The latter term relates to "channel binding" [RFC5056]. Connection latching is suitable for use in channel binding applications, or will be, at any rate, when the channel bindings for IPsec channels are defined (the specification of IPsec channel bindings is out of scope for this document).

1.1. Conventions used in this document

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].

2. Connection Latching

An "IPsec channel" is a packet flow associated with a ULP control block, such as a TCP connection, where all the packets are protected by IPsec SAs such that:

- o the peer's identity is the same for the lifetime of the packet flow

- o the quality of IPsec protection used for the packet flow's individual packets is the same for all of them for the lifetime of the packet flow

An IPsec channel is created when the associated packet flow is created. This can be the result of a local operation (e.g., a connect()) that causes the initial outgoing packet for that flow to be sent, or it can be the result of receiving the first/initiating packet for that flow (e.g., a TCP SYN packet).

IPsec channels are created by "latching" various parameters listed below to a ULP connection when the connections are created. The REQUIRED set of parameters bound in IPsec channels is:

- o Type of protection: confidentiality and/or integrity protection;

- o Transport mode vs. tunnel mode;

- o Quality of protection: cryptographic algorithm suites, key lengths, and replay protection;

- o Peer identity: peers' asserted and authorized IDs, as per the IPsec processing model [RFC4301] and BTNS [I-D.ietf-btns-core].

Additionally, there SHOULD be an optional way for applications to specify the conflict resolution behaviour of an IPsec channel (see description of the SUSPENDED and BROKEN connection latch states in Section 2.1): whether to wait for the conflict to disappear, or whether to break the channel. The default for this option SHOULD be "break the channel", and MAY be configurable through local policy.

The SAs that protect an IPsec channel's packets need not be related by anything other than the fact that they must be congruent to the channel (i.e, the SAs' parameters must match those that are latched into the channel). In particular, it is desirable that IPsec channels survive the expiration of IKE_SAs and child SAs -- new ones can be negotiated as necessary without compromising the security guarantees of the channel -- because operational considerations of the various key exchange protocols then cannot affect the design and

features of connection latching.

Implementations SHOULD provide applications with APIs for inquiring whether a connection is latched and what the latched parameters are. Implementations SHOULD provide applications with some control, through application programming interfaces (APIs) [I-D.ietf-btns-abstract-api], over what quality of protection, or the expected identity of a peer. If an application does not use such interfaces then it will obtain default quality of protection derived from system policy. Implementations MAY create IPsec channels automatically by default when the application does not request an IPsec channel.

Requirements and recommendations:

- If an IPsec channel is desired then packets for a given connection MUST NOT be sent until the channel is established.

- If an IPsec channel is desired then inbound packets for a given connection MUST NOT be accepted until the channel is established. I.e., inbound packets for a given connection arriving prior to the establishment of the corresponding IPsec channel must be dropped or the channel establishment must fail.

- Once an IPsec channel is established packets for the latched connection MUST NOT be sent unprotected nor protected by an SA that does not match the latched parameters.

- Once an IPsec channel is established packets for the latched connection MUST NOT be accepted unprotected nor protected by an SA that does not match the latched parameters. I.e., such packets either must be dropped or must cause the channel to be terminated and the application to be informed before data from such a packet can be delivered to the application.

- Native implementations SHOULD provide programming interfaces for inquiring the values of the parameters latched in a connection.

- Implementations that provide such programming interfaces MUST make available to applications all relevant information about a peer's ID, including authentication information. This includes the peer certificate, when one is used, and the trust anchor that it was validated to.

- Implementations that provide such programming interfaces SHOULD make available to applications any available NAT-related information about the peer: whether it is behind a NAT and, if it is, the inner and outer tunnel addresses of the peer.

- o Native implementations SHOULD provide programming interfaces for setting the values of the parameters to be latched in a connection that will be initiated or accepted, but these interfaces MUST limit what values applications may request according to system policy (i.e., the IPsec PAD and SPD) and the application's privilege.

  (Typical system policy may not allow applications any freedom here. Policy extensions allowing for optional protection are described in Section 3.)

- o The parameters latched in an IPsec channel MUST remain unchanged once the channel is established.

- o Timeouts while establishing an SA with parameters that match a those latched into an IPsec channel MUST be treated as packet loss (as happens, for example, when a network partitions); normal ULP and/or application timeout handling and retransmission considerations apply. Failure to establish an appropriate SA for an IPsec channel SHOULD be communicated to the ULP and application, and MAY cause the IPsec channel to be broken (which MUST be communicated to the ULP and application).

- o Implementations that have a restartable key management process (or "daemon") MUST arrange for existing latched connections to either be broken and disconnected, or for them to survive the restart of key exchange processes. (This is implied by the above requirements.) For example, if such an implementation relies on keeping some aspects of connection latch state in the restartable key management process (e.g., potentially large values, such as BTNS peer IDs), then either such state must be restored on restart of such a process, or outstanding connection latches must be transitioned to the CLOSED state.

- o Dynamic IPsec policy related to connection latches MUST be torn down when latched connections are torn down, even when the latter is implied, such as at crash/halt/reboot time.

We describe two models (one normative) of IPsec channels for native IPsec implementations. Both models should suffice for all-software native implementations of IPsec. One, the other or both models should be workable for most native implementations where part of the IPsec stack is implemented in hardware. The normative model is based on abstract programming interfaces between ULPs and the key management component of IPsec. The second model is based on abstract programming interfaces between ULPs and the IPsec (ESP/AH) layer in the IP stack.

The two models given below are not, however, entirely equivalent. One model cannot be implemented with NICs that offload ESP/AH but which do not tag incoming packets passed to the host processor with SA information, nor allow the host processor to so tag outgoing packets. That same model can be extended to support connection latching with unconnected datagram sockets, while the other model cannot be so extended. There may be other minor differences between the two models; rather than seek to prove equivalency for some set of security guarantees we instead choose one model to be the normative one.

We also provide a model for non-native implementations, such as bump-in-the-stack (BITS) and SG implementations. The connection latching model for non-native implementations is not full-featured as it depends on estimating packet flow state, which may not always be possible. Nor can non-native IPsec implementations be expected to provide APIs related to connection latching (implementations that do could be said to be native). As such this third model is not suitable for channel binding applications [RFC5056].

2.1. Connection latch state machine

Connection latches can exist in any of the following five states:

- o LISTENER
- o ESTABLISHED
- o SUSPENDED (there exist conflicting SAs, waiting for them to expire or be removed)
- o BROKEN (conflicting SAs were created)
- o CLOSED (by the ULP, the application or administratively)

and always have an associated owner, or holder, such as a ULP transmission control block (TCB).

A connection latch can be born in the LISTENER state, which can transition only to the CLOSED state. The LISTENER state corresponds to LISTEN state of TCP sockets and is associated with IP 3-tuples, and can give rise to new connection latches in the ESTABLISHED state.

A connection latch can also be born in the ESTABLISHED state, either through the direct initiative of a ULP or when an event occurs that causes a LISTENER latch to create an ESTABLISHED latch. This state represents an active connection latch for a traffic flow's 5-tuple. ESTABLISHED connection latches can transition to the SUSPENDED,

BROKEN, and CLOSED states.

Connection latches remain in the CLOSED state until their owners are informed except where the ownser caused the transition, in which case this state is fleeting. Transitions to the CLOSED state should typically be initiated by latch owners, but implementations MAY provide administrative interfaces through which to close active latches.

Connection latches transition to either the SUSPENDED or BROKEN states, according to application preference (or system policy), when there exist SAs in the SAD whose traffic selectors encompass the 5-tuple bound by the latch, and whose peer and/or parameters conflict with those bound by the latch. Transitions to the SUSPENDED always cause the associated owner to be informed. Connection latches in the SUSPENDED state may transition back to ESTABLISHED when the conflict is cleared. Transitions to either state always cause the associated owner to be notified. BROKEN connection latches can only transition to CLOSED, but SUSPENDED latches can transition to either ESTABLISHED or CLOSED (see above).

Most state transitions are the result of local actions of the latch owners (ULPs). The only exceptions are: birth into the ESTABLISHED state from a LISTENER latch, transitions to the SUSPENDED and BROKEN states, and administrative transitions to the CLOSED state. (Additionally, see the implementation note about restartable key management processes in Section 2.)

The details of the transitions depend on the model of connection latching followed by any given implementation. See the following sections.

2.2. Normative Model: ULP interfaces to the key manager

This section is NORMATIVE.

In this section we describe connection latching in terms of an interface between ULPs and the key manager component of a native IPsec implementation. Abstract interfaces for creating, inquiring about, and releasing IPsec channels are described.

This model adds a service to the IPsec key manager (i.e., the component that manages the SAD and interfaces with, or implements, key exchange protocols): management of connection latches. There is also a new IPsec database, the Latch Database (LD), that contains all connection latch objects.

The traditional IPsec processing model allows the concurrent

existence of SAs with different peers but overlapping traffic selectors. Such behaviour, in this model, directly violates the requirements for connection latching. We address this problem by requiring that connection latches be broken (and holders informed) when such conflicts arise.

The ULP interfaces to the IPsec PAD database are as follows:

o CREATE_LISTENER_LATCH(3-tuple, [type and quality of protection parameters]) -> latch handle

   If there is no conflicting connection latch object in the LISTENER state for the given 3-tuple (local address, protocol and local port number), and local policy permits it, then this operation atomically creates a connection latch object in the LISTENER state for the given 3-tuple.

   When a child SA is negotiated that would match a listener latch's 3-tuple then the key manager SHOULD narrow the child SA so that its local address and port ranges do not include the 3-tuple or so that the SA has only one local address and port number: the one from the tuple.

   When a child SA is created that matches a listener latch's 3-tuple, but not any ESTABLISHED connection latch's 5-tuple (local address, remote address, protocol, local port number and remote port number), then the key manager creates a new connection latch object in the ESTABLISHED state. The key manager MUST inform the holder of the listener latch of connection latches created as a result of the listener latch.

o CREATE_CONNECTION_LATCH(5-tuple, [type and quality of protection parameters], [peer ID], [local ID]) -> latch handle

   If no connection latch exists in the ESTABLISHED states with the same 5-tuple, and if there exist no child SAs that match the given 5-tuple, or all such SAs share the same type and quality of protection parameters and the same peer then this operation creates a connection latch object in the ESTABLISHED state for the given 5-tuple. If the caller provided all the optional arguments to this operation then the resulting connection latch can be created in the ESTABLISHED state directly.

   If there exist no child SAs matching the given 5-tuple then the key manager SHOULD try to create a pair of child SAs for that 5-tuple. In any case, the key manager can expect that the ULP will send a packet that would trigger the creation of such SAs.

When the key manager tries to create child SAs it should narrow the proposals so that their traffic selector match no connection latches in the ESTABLISHED states, or so that they match only the 5-tuple of a single such connection latch.

o RELEASE_LATCH(latch object handle)

   Changes the state of the given connection latch to CLOSED; the connection latch is then deleted.

   The key manager SHOULD delete any existing child SAs that match the given latch if it had been in the ESTABLISHED states. If the key manager does delete such SAs then it SHOULD inform the peer with an informational Delete payload (see IKEv2 [RFC4306]).

o INQUIRE_LATCH(latch object handle) -> latch state, latched parameters

   Returns all available information about the given latch.

Needless to say, the LD is updated whenever a connection latch object is created, deleted or broken.

The API described above is a new service of the IPsec key manager. In particular the IPsec key manager MUST prevent conflicts amongst latches, and it MUST prevent conflicts between any latch and existing or proposed child SAs as follows:

o Non-listener connection latches MUST NOT be created if there exist conflicting SAs in the SAD at the time the connection latch is requested or would be created (from a listener latch). A child SA conflicts with another, in view of a latch, if and only if: a) its traffic selectors and the conflicting SA's match the give latch's, b) its peer, type of protection, or quality of protection parameters differ from the conflicting SA.

o Child SA proposals that would conflict with an extant connection latch and whose traffic selectors can be narrowed to avoid the conflict MUST be narrowed (see section 2.9 of [RFC4306]);

o Where child SA proposals that would conflict with an extant connection latch cannot be narrowed to avoid the conflict the key manager MUST break the connection latch and inform the holder (i.e., the ULP) prior to accepting the conflicting SAs.

Additionally, the key manager MUST protect latched connections against SPD changes that would change the quality of protection

afforded to a latched connection's traffic, or which would bypass it. When such a configuration change takes place the key manager MUST either preserve a logical SPD entry such that the latched connection continues to obtain the required protection, or the key manager MUST break the latch and inform the latch holder (ULP) before the change takes place. To do this the key manager can logically update the SPD as if a PROTECT entry had been added at the head of the SPD-S with traffic selectors matching only the latched connection's 5-tuple, and with processing information taken from the actual SPD entry matched by the connection (possibly augmented by the application's request for additional protection). Such updates of the SPD MUST NOT survive system crashes or reboots.

ULPs create latched connections by interfacing with IPsec below as follows:

- For listening end-points the ULP will request a connection latch listener object for the ULP listener's 3-tuple. Any latching parameters requested by the application should be passed along.

- When the ULP receives a packet initiating a connection for a 5-tuple matching a 3-tuple listener latch, then the ULP will ask the key manager whether a 5-tuple connection latch was created. If not then the ULP will either reject the new connection or accept it and inform the application that the new connection is not latched (that it does not represent an IPsec channel).

- When initiating a connection the ULP will request a connection latch object for the connection's 5-tuple. Any latching parameters requested by the application should be passed along. If no latch can be created then the ULP will either return an error to the application or continue with the new connection and inform the application that the new connection is not latched.

- When a latched connection is torn down and no further packets are expected for it then the ULP will request that the connection latch object be destroyed.

- When tearing down a listener the ULP will request that the connection latch listener object be destroyed.

- When a ULP listener rejects connections the ULP will request the destruction of any connection latch objects that may have been created as a result of the peer's attempt to open the connection.

- When the key manager informs a ULP that a connection latch is no longer valid then the ULP SHOULD reset or otherwise terminate the connection and MUST inform the application.

The main benefit of this model of connection latching is that it accommodates IPsec implementations where ESP/AH handling is implemented in hardware (for all or a subset of the host's SAD), but where the hardware does not support tagging inbound packets with the indexes of SAD entries corresponding to the SAs that protected them.

Note that there is a race condition in this method of connection latching: incoming packets may race with the ULP and the IPsec key manager's manipulation of connection latch objects and SAD entries. As a result ULPs may not be able to trust some packets even though a suitable connection latch object may exist. Implementations MUST prevent such races. One method to prevent these races is to tag packets passed up by the ESP/AH layer with a key manager state version number that is monotonically incremented every time that connection latching state changes; this version number must be incremented atomically relative to the SAD and the LD, including SAD subsets stored on IPsec offload hardware. Other methods may be possible, including dropping packets that arrive within a certain amount of time since the creation/destruction of connection latch objects (e.g., if the maximum latency within the key manager and IP stack is known and guaranteed).

2.3. Informative model: local packet tagging

In this section we describe connection latching in terms of interfaces between ULPs and IPsec based on tagging packets as they go up and down the IP stack.

This section is INFORMATIVE.

The ULPs and IPsec interface through a local packet tagging scheme (i.e., the tags don't appear on the wire):

- The IPsec layer tags all inbound protected packets addressed to the host with the index of the SAD entry corresponding to the SA that protected the packet.

- The IPsec layer understands two types of tags on outbound packets:

   * a tag specifying a set of latched parameters (peer ID, quality of protection, etc...) that the IPsec layer will use to find or acquire an appropriate SA for protecting the outbound packet (else IPsec will inform the ULP and drop the packet);

   * a tag requesting feedback about the SA used to protect the outgoing packet, if any.

ULPs create latched connections by interfacing with IPsec below as

follows:

- o When the ULP passes a connection's initiating packet to IP the ULP requests feedback about the SA used to protect the outgoing packet, if any, and may specify latching parameters requested by the application. If the packet is protected by IPsec then the ULP records certain parameters of the SA used to protect it in the connection's TCB.

- o When a ULP receives a connection's initiating packet it processes the IPsec tag of the packet, and it records in the connection's TCB the parameters of the SA that should be latched.

Once SA parameters are recorded in a connection's TCB the ULP enforces the connection's latch, or binding, to these parameters as follows:

- o The ULP processes the IPsec tag of all inbound packets for a given connection and checks that the SAs used to protect input packets match the connection latches recorded in the TCBs. Packets which are not so protected are dropped (this corresponds to transitioning the connection latch to the SUSPENDED state until the next acceptable packet arrives, but in this model this transition is imaginary), or cause the ULP to break the connection latch and inform the application.

- o The ULP always requests that outgoing packets be protected by SAs that match the latched connection by appropriately tagging outbound packets.

The receipt of a packet matching a latched connection's 5-tuple, but protected by an SA with an inappropriate peer, SHOULD be taken as an indication that the original peer is no longer at the original address and that the connection SHOULD be reset, the application informed, and the connection latch removed.

This model of connection latching may not be workable with ESP/AH offload hardware that does not support the packet tagging scheme described above.

Extending the ULP/IPsec interface to the application should enable applications to use connection-less datagram transports and implement connection latching at the application layer.

2.4. Non-native mode IPsec

Non-native IPsec implementations, primarily BITS and SG, can implement connection latching too. One major distinction between

native IPsec and BITS/BITW/SG IPsec is the lack of APIs for applications at the end-points in the case of the latter. As a result there can be no uses of the latch management interfaces as described in Section 2.2, not at the ULP end-points. Therefore BITS/BITW/SG implementations must discern ULP connection state from packet inspection (which many firewalls can do) and emulate calls to the key manager accordingly.

When a connection latch is broken a BITS/BITW/SG implementation may have to fake a connection reset by sending appropriate packets (e.g., TCP RST packets), for the affected connections.

As with all stateful middle-boxes this scheme suffers from the inability of the middle-box to interact with the applications. For example, connection death may be difficult to ascertain. Nor can channel binding applications work with channels maintained by proxy without being able to communicate (securely) about it with the middle-box.

2.5. Conflict Resolution

Consider a system, say, an IMAP server, with an IPsec policy allowing all peers with certificates issued by some CA to claim any dynamically allocated address in a local network.

In such an environment a peer might appear using some address, then disappear (e.g., a laptop whose battery runs out) and another peer might later (after the first peer's DHCP lease expires) appear using the same IP address as the first peer. The first peer might have had a long-lived TCP connection open with the server. The new peer might try to open a connection with the same server and with the same 5-tuple as the first peer. The new peer's TCP SYN packet will fail to match the existing connection's latch.

In such cases implementations based on Section 2.2 and Section 2.4 will be unable to narrow the new peer's child SA proposals to avoid a conflict, and must either reject them (and transition the existing latch to SUSPENDED) or terminate the existing connection latch (i.e., transition it to the BROKEN state).

Implementors MUST provide termination of the existing connection as the default behaviour in such cases. Implementors MAY provide a configuration option for selecting the other behaviours.

3. Optional protection

Given IPsec APIs an application could request that a connection's packets be protected where they would otherwise be bypassed; that is, applications could override BYPASS policy. Locally privileged applications could request that their connections' packets be bypassed rather than protected; that is, privileged applications could override PROTECT policy. We call this "optional protection."

Both native IPsec models of connection latching can be extended to support optional protection. With the model described in Section 2.3 optional protection comes naturally: the IPsec layer need only check that the protection requested for outbound packets meets or exceeds (as determined by local or system policy) the quality of protection, if any, required by the SPD. Similarly, for the model described in Section 2.2 the check that requested protection meets or exceeds that required by the SPD is performed by the IPsec key manager when creating connection latch and connection latch listener objects.

When an application requests, and IPsec permits, either additional protection, or bypassing protection, then the SPD MUST be logically updated such that there exists a suitable SPD entry protecting or bypassing the exact 5-tuple recorded by the corresponding connection latch. Such logical SPD updates MUST be made at connection latch creation time, and MUST be made atomically (see the note about race conditions in Section 2.2). Such updates of the SPD MUST NOT survive system crashes or reboots.

4. Simulataneous latch establishment

Some connection-oriented ULPs, specifically TCP, support simulaneous connections (where two clients connect to each other, using the same 5-tuple, at the same time). Connection latching supports simultaneous latching as well, provided that the key exchange protocol does not make it impossible.

Consider two applications doing a simultaneous TCP connect to each other and requesting an IPsec channel. If they request the same connection latching parameters, then the connection and channel should be established as usual. Even if the key exchange protocol in use doesn't support simultaneous IKE_SA and/or child SA establishment, provided one peer's attempt to create the necessary child SAs succeeds then the other peer should be able to notice the new SAs immediately upon failure of its attempts to create the same.

If, however, the two peer applications were to request different connection latching parameters, then the connection latch must fail on one end (if the key exchange protocol does not support simultaneous SA creation) or on both ends.

5. Security Considerations

Connection latching protects only individual connections from weak peer ID<->address binding, IPsec configuration changes, and from configurations that allow multiple peers to assert the same addresses. But connection latching does not ensure that any two connections with the same end-point addresses will have the same latched peer IDs. In other words, applications that use multiple concurrent connections between two given nodes are not protected any more or less by use of IPsec connection latching than by use of IPsec alone. Such multi-connection applications can, however, examine the latched SA parameters of each connection to ensure that all concurrent connections with the same end-point addresses also have the same end-point IPsec IDs.

Applications which are sensitive to connection closure, such as the Border Gateway Protocol (BGP), SHOULD set the conflict resolution option for connection latching (e.g., in the case of BGP that option should be set to "wait for the conflict to be resolved").

IPsec channels are a pre-requisite for channel binding [RFC5056] to IPsec. Connection latching provides such channels, but the process of binding IPsec channels (latched connections) to authentication at application layers is not specified herein.

Without IPsec APIs connection latching provides marginal security benefits over traditional IPsec. Such APIs are not described herein; see [I-D.ietf-btns-abstract-api].

6. IANA Considerations

There are not IANA considerations for this document.

7. Acknowledgements

The author thanks Michael Richardson for all his help, as well as Stephen Kent, Sam Hartman, Bill Sommerfeld, Dan McDonald, and many others who've participated in the BTNS WG or who've answered questions about IPsec, connection latching implementations, etc...

8. References

8.1. Normative References

[I-D.ietf-btns-core]
Williams, N. and M. Richardson, "Better-Than-Nothing-Security: An Unauthenticated Mode of IPsec", draft-ietf-btns-core-06 (work in progress), January 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-06 (work in progress), October 2007.

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

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

[RFC4306] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol", RFC 4306, December 2005.

8.2. Informative References

[I-D.bellovin-useipsec]
Bellovin, S., "Guidelines for Mandating the Use of IPsec Version 2", draft-bellovin-useipsec-07 (work in progress), October 2007.

[I-D.dondeti-useipsec-430x]
Dondeti, L. and V. Narayanan, "Guidelines for using IPsec and IKEv2", draft-dondeti-useipsec-430x-00 (work in progress), October 2006.

[I-D.ietf-btns-abstract-api]
Richardson, M., "An interface between applications and keying systems", draft-ietf-btns-abstract-api-00 (work in progress), June 2007.

[IP_SEC_OPT.man]
Sun Microsystems, Inc., "Solaris ipsec(7P) manpage", October 2006.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.

[RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, November 2007.

Author's Address

Nicolas Williams
Sun Microsystems
5300 Riata Trace Ct
Austin, TX 78727
US

Email: Nicolas.Williams@sun.com

Full Copyright Statement

Intellectual Property

Acknowledgment

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