NETWORK WORKING GROUP
Internet-Draft
Expires: May 7, 2009

N. Williams
Sun

November 3, 2008

IPsec Channels: Connection Latching
draft-ietf-btns-connection-latching-08.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. Connection latching is layered on top of IPsec and does not modify the underlying IPsec architecture.

Connection latching 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.

Finally, the availability of IPsec channels will make it possible to use channel binding to IPsec channels.

Table of Contents

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 Security Policy Database (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 [RFC0768], TCP [RFC0793] and SCTP [RFC4960] 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.

Connection latching, as specified herein, is primarily about watching updates to the SPD and Security Association Database (SAD) to detect changes that are adverse to an application's requirements for any given packet flow, and to react accordingly (such as by synchronously alerting the ULP and application before packets can be sent or received under the new policy). Under no circumstance are IPsec

policy databases to be modified by connection latching in any way that can persist beyond the lifetime of the related packet flows, nor reboots. Under no circumstance is the PAD to be modified at all by connection latching. If all optional features of connection latching are excluded then connection latching can be implemented as a monitor of SPD and SAD changes that intrudes in their workings no more than is needed to provide synchronous alerts to ULPs and applications.

We assume the reader is familiar with the IPsec architecture [RFC4301] and IKEv2 [RFC4306].

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).

Note: where this document mentions IPsec peer "ID" it refers to the IKE peer ID (e.g., the ID derived from a peer's cert, as well as the cert), not the peer's IP address.

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].

Abstract function names are all capitalized and denoted by a pair of parenthesis. In their descriptions the arguments appear within the parenthesis, with optional argument surrounded by square brackets. Return values, if any, are indicated by following the function argument list with "->" and a description of the return value. For example, "FOO(3-tuple, [message])" would be a function named "FOO" with two arguments, one of them optional, and returning nothing, whereas "FOOBAR(handle) -> state" would be a function with a single, required argument that returns a value. The values' types are described in the surrounding text.

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:

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

An IPsec channel is destroyed when the associated packet flow ends. An IPsec channel can also be "broken" when the connection latch cannot be maintained for some reason (see below), in which case the ULP and application are informed.

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:

- Type of protection: confidentiality and/or integrity protection;
- Transport mode vs. tunnel mode;
- Quality of protection: cryptographic algorithm suites, key lengths, and replay protection;
- Local identity: the local ID asserted to the peer, as per the IPsec processing model [RFC4301] and BTNS [I-D.ietf-btns-core];
- Peer identity: the peer's asserted and authorized IDs, as per the IPsec processing model [RFC4301] and BTNS [I-D.ietf-btns-core].

The SAs that protect a given IPsec channel's packets may change over time in that they may expire and be replaced with equivalent SAs, or may be re-keyed. The set 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 because operational considerations of the various key exchange protocols then cannot affect the design and features of connection latching.

When a situation arises in which the SPD is modified, or an SA is added to the SAD, such that the new policy or SA are not congruent to an established channel (see previous paragraph) then we consider this a conflict. Conflict resolution is addressed below.

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.
- 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 and available 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 (but not necessarily the whole certificate validation chain).
- 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.
- 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 local privileges.

  (Typical system policy may not allow applications any choices here. Policy extensions allowing for optional protection are described in Section 3.1.)
- Implementations SHOULD create IPsec channels automatically by default when the application does not explicitly request an IPsec channel.
- The parameters latched in an IPsec channel MUST remain unchanged once the channel is established.
- Timeouts while establishing child SAs with parameters that match 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.
- 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.

- Dynamic IPsec policy (see Section 3.1) related to connection latches, if any, MUST be torn down when latched connections are torn down, and MUST NOT survive reboots.

We describe two models, one of them normative, of IPsec channels for native IPsec implementations. The normative model is based on abstract programming interfaces in the form of function calls between ULPs and the key management component of IPsec (basically, the SAD, augmented with a Latch Database (LD)). The second model is based on abstract programming interfaces between ULPs and the IPsec (ESP/AH) layer in the form of meta-data tagging of packets within 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 easily extended to support connection latching with unconnected datagram sockets, while the other model is rigidly tied to a notion of "connections" and cannot be so extended. There may be other minor differences between the two models. Rather than seek to establish 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:

- LISTENER
- ESTABLISHED
- BROKEN (there exist SAs which conflict with the given connection latch)
- 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 (and other) 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 and BROKEN states, either through the direct initiative of a ULP or when an event occurs that causes a LISTENER latch to create a new latch (in either ESTABLISHED or BROKEN states). These states represents an active connection latch for a traffic flow's 5-tuple. Connection latches in these two states can transition to the other of the two states, as well as to the CLOSED state.

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 from ESTABLISHED or BROKEN states to the CLOSED state should typically be initiated by latch owners, but implementations SHOULD provide administrative interfaces through which to close active latches.

Connection latches transition to the BROKEN state 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 BROKEN state always cause the associated owner to be informed. Connection latches in the BROKEN state transition back to ESTABLISHED when the conflict is cleared.

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

The state diagram below makes use of conventions described in Section 1.1 and state transition events described in Section 2.2.

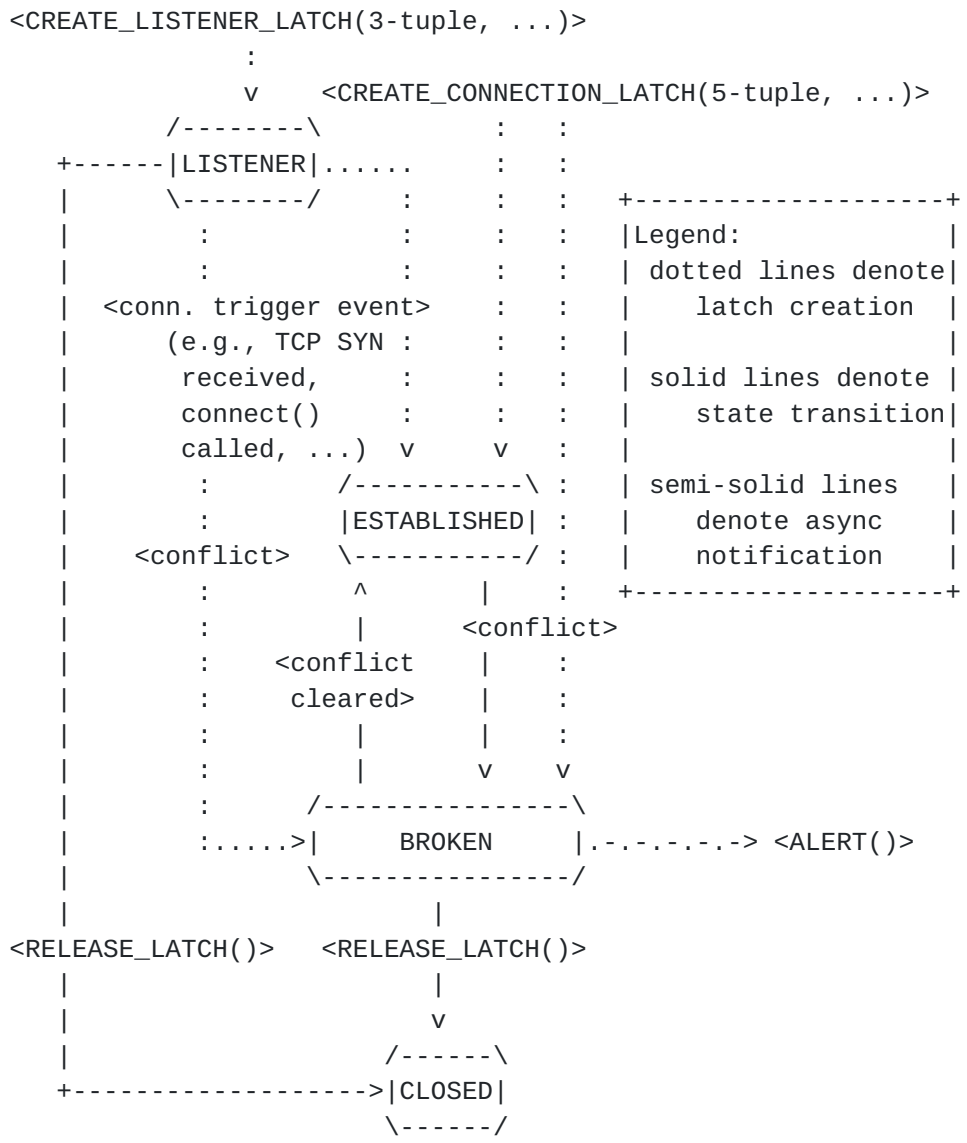

Figure 1: Connection Latching State Machine

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 describes the NORMATIVE model of connection latching.

In this section we describe connection latching in terms of a function call 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 separate implementations of, or directly 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 LD does not persist across system reboots.

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 (see Section 2). We address this problem by requiring that connection latches be broken (and holders informed) when such conflicts arise.

The following INFORMATIVE figure illustrates this model and API in terms that are familiar to many implementors, though not applicable to all:

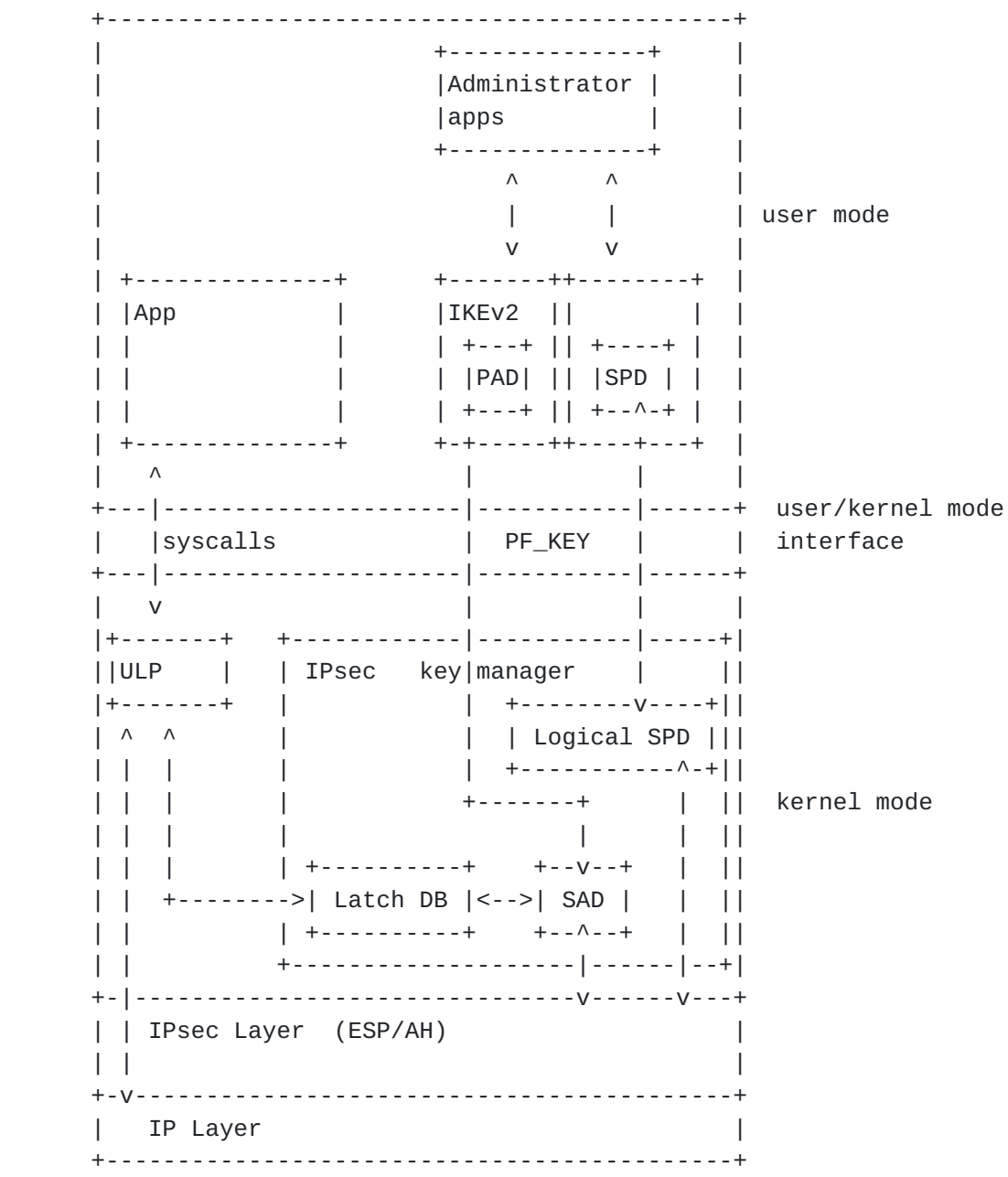

Figure 2: Informative Implementation Architecture Diagram

The ULP interfaces to the IPsec LD 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 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.

- 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, then the system MUST initiate an IKE exchange to setup child SAs for this 5-tuple. Once one or more applicable SAs exist in the SAD and 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.
- RELEASE_LATCH(latch object handle)

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

  The key manager MAY 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]).
- FIND_LATCH(5-tuple) -> latch object handle

  Given a 5-tuple returns a latch handle (or an error).
- INQUIRE_LATCH(latch object handle) -> latch state, latched parameters

  Returns all available information about the given latch, including its current state (or an error).

The IPsec LD interface to the ULP is as follows:

- ALERT(latch object handle, 5-tuple, new state, [reason])

  Alerts a ULP as to an asynchronous state change for the given connection latch and, optionally, provides a reason for the change.

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:

- 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 given latch's, and b) its peer, type of protection, or quality of protection parameters differ from the conflicting SA.
- Child SA proposals that would conflict with an extant connection latch and whose traffic selectors can be narrowed to avoid the conflict SHOULD be narrowed (see section 2.9 of [RFC4306]), otherwise the latch MUST be transitioned to the BROKEN state.
- 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.

Finally, 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 respond in either of the following ways. The REQUIRED to implement behaviour is to transition into the BROKEN state all connection latches that conflict with the given SPD change. An OPTIONAL behaviour is to 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 connection latch. 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 MUST 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.
- 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 MUST be passed along. If no latch can be created then the ULP MUST 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 connection is torn down and no further packets are expected for it then the ULP MUST request that the connection latch object be destroyed.
- When tearing down a listener the ULP MUST 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 has transitioned to the BROKEN state then the ULP MUST stop sending packets and MUST drop all subsequent incoming packets for the affected connection until it transitions back to ESTABLISHED. Connection-oriented ULPs SHOULD act as though the connection is experiencing packet loss.
- When the key manager informs a ULP that a connection latch has been administratively transitioned to the CLOSED state then connection-oriented ULPs MUST act as though the connection has been reset by the peer. Implementations of ULPs which are not connection-oriented, and which have no API by which to simulate a reset, MUST drop all inbound packets for that connection and MUST NOT send any further packets -- the application is expected to detect timeouts and act accordingly.

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.

2.2.1. Race Conditions and Corner Cases

ULPs MUST drop inbound packets and stop sending packets immediately upon receipt of a connection latch break message. Otherwise the ULP will not be able to distinguish inbound packets that were protected consistently with the connection's latch from inbound packets that were not. This may include dropping inbound packets that were protected by a suitable SA; dropping such packets is no different, from the ULP's point of view, than packet loss elsewhere on the network at the IP layer or below -- harmless, from a security point of view as the connection fails safe, but it can result in retransmits.

Another race condition is as follows. A PROTECTed TCP SYN packet may be received and decapsulated but the SA that protected it could expired before the key manager creates the connection latch that would be created by that packet. In this case the key manager will

have to initiate new child SAs so as to determine what the sender's peer ID is so it can be included in the connection latch. Here there is no guarantee that the peer ID for the new SAs will be the same as those of the peer that sent the TCP SYN packet. This race condition is harmless: TCP will send a SYN+ACK to the wrong peer, which will then respond with an RST -- the connection latch will reflect the new peer however, so if the new peer is malicious it will not be able to appear to be the old peer. Therefore this race condition is harmless.

2.2.2. Example

Consider several systems with a very simple PAD containing a single entry like so:

Rule	Remote ID	Child SA IDs allowed	SPD Search by
1	<any valid to trust anchor X> 10/8		by-IP

Figure 3: Example PAD

And a simple SPD like so:

Rule	Local TS	Remote TS	Next Proto	Action
1	10/8:ANY	10/8:1-5000	TCP	PROTECT(ESP,...)
1	10/8:1-5000	10/8:ANY	TCP	PROTECT(ESP,...)
1	ANY	ANY	ANY	BYPASS

Figure 4: [SG-A] SPD table

Effectively this says: for TCP ports 1-5000 in our network allow only peers that have credentials issued by CA X and PROTECT that traffic with ESP, otherwise bypass all other traffic.

Now let's consider two hosts, A and B, in this network that wish to communicate using port 4000, and a third host, C, that is also in the same network and wishes to attack A and/or B. All three hosts have credentials and certificates issued by CA X. Let's also imagine that A is connected to its network via a wireless link and is dynamically address.

B is listening on port 4000. A initiates a connection from port 32800 to B on port 4000.

We'll assume no IPsec APIs, but that TCP creates latches where possible.

We'll consider three cases: a) A and B both support connection latching, b) only A does, c) only B does. For the purposes of this example the SAD is empty on all three hosts when A initiates its TCP connection to B on port 4000.

When an application running on A initiates a TCP connection to B on port 4000, A will begin by creating a connection latch. Since the SAD is empty A will initiate an IKEv2 exchange to create an IKE_SA with B and a pair of CHILD SAs for the 5-tuple {TCP, A, 32800, B, 4000}, then a new latch will be created in ESTABLISHED state. Sometime later TCP will send a SYN packet protected by the A-to-B child SA, per the SPD.

When an application running on B creates a TCP listener socket on port 4000, B will create a LISTENER connection latch for the 3-tuple {TCP, B, 4000}. When B receives A's TCP SYN packet it will then create a connection latch for {TCP, B, 4000, A, 32800}. Since by this point CHILD SAs have been created whose traffic selectors encompass this 5-tuple and there are no other conflicting SAs in the SAD, this connection latch will be created in the ESTABLISHED state.

If C attempts to mount a man-in-the-middle attack on A (i.e., pretend to have B's address(es)) any time after A created its connection latch then C's SAs with A will cause the connection latch to break, and the TCP connection to be reset (since we assume no APIs by which TCP could notify the application of the connection latch break). If C attempts to impersonate A to B then the same thing will happen on B.

If A does not support connection latching then C will be able to impersonate B to A at any time, but without having seen the cleartext of traffic between A and B, C will be limited by the TCP sequence numbers to attacks such as RST attacks. Similarly if B does not support connection latching.

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.

In this model the ULPs maintain connection latch objects and state, rather than the IPsec key manager, as well as effectively caching a

subset of the decorrelated SPD in ULP TCBs. Local tagging of packets as they move up and down the stack with SA identifiers then allows the ULPs to enforce connection latching semantics. These tags, of course, don't appear on the wire.

The interface between the ULPs and IPsec interface is as follows:

- 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:

- 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.
- 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:

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

By effectively caching a subset of the decorrelated SPD in ULP TCBs and through its packet tagging nature, this method of connection latching can also optimize processing of the SPD by obviating the need to search it, both, on input and output, for packets intended

for the host or originated by the host. This makes implementation of the OPTIONAL "logical SPD" updates described in Section 2.2 and Section 3.1 an incidental side effect of this approach.

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

Note that this model has no BROKEN connection latch state.

Extending the ULP/IPsec packet-tagging interface to the application for use with connection-less datagram transports should enable applications to use such 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. Implementation Note Regarding Peer IDs

One of the recommendations for connection latching implementators is to make available peer CERT payloads (certificates) to the applications.

Additionally, raw public keys are likely to be used in the construction of channel bindings for IPsec channels; see [I-D.williams-ipsec-channel-binding], and must be available, in any case, in order to implement leap-of-faith at the application layer

(see [I-D.ietf-btns-core] and [I-D.ietf-btns-prob-and-applic]).

Certificates and raw public keys are large bit strings, too large to be reasonably kept in kernel-mode implementations of connection latching (which will likely be the typical case). Such implementations should intern peer IDs in a user-mode database and use small integers to refer to them from the kernel-mode SAD and LD. Corruption of such a database is akin to corruption of the SAD/LD; in the event of corruption the implementation MUST act as though all ESTABLISHED and BROKEN connection latches are administratively transitioned to the CLOSED state. Implemenations without IPsec APIs MAY hash peer IDs and use the hash to refer to them, preferably using a strong hash algorithm.

3. Optional Features

At its bare minimum connection latching is a passive layer atop IPsec that warn ULPs of SPD and SAD changes that are incompatible with the SPD/SAD state that was applicable to a connection when it was established.

There are some optional features, such as (abstract) APIs. Some of these features make connection latching a somewhat more active feature. Specifically, the optional logical SPD updates described in Section 2.2 and the optional protection/bypass feature described in the following sub-section.

3.1. 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 local policy 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 or on both ends.

5. Connection Latching to IPsec for Various ULPs

The following sub-sections describe connection latching for each of three transport protocols. Note that for TCP and UDP there is nothing in the following sections that should not already be obvious from the remainder of this document. The section on SCTP, however, specifies details related to SCTP multi-homing, that may not be as obvious.

5.1. Connection Latching to IPsec for TCP

IPsec connection latch creation/release for TCP [RFC0793] connections is triggered when:

- A TCP listener end-point is created (e.g., when the BSD sockets listen() function is called on a socket). This should cause the creation of a LISTENER connection latch.
- A TCP SYN packet is received on an IP address and port number for

which there is a listener. This should cause the creation of an ESTABLISHED or BROKEN connection latch.

- o A TCP SYN packet is sent (e.g., as the result of a call to the BSD sockets connect() function). This should cause the creation of an ESTABLISHED or BROKEN connection latch.
- o Any state transition of a TCP connection to the CLOSED state will cause a corresponding transition for any associated connection latch to the CLOSED state as well.

When a connection latch transitions to the BROKEN state and the application requested (or system policy dictates it) that the connection be broken, then TCP should inform the application, if there is a way to do so, or else it should wait, allowing the TCP keepalive (or application-layer keepalive strategy) to timeout the connection, if enabled. When a connection latch is administratively transitioned to the CLOSED state then TCP should act as though an RST packet has been received.

5.2. Connection Latching to IPsec for UDP with Simulated Connections

UDP [RFC0768] is a connection-less transport protocol. However, some networking APIs (e.g., the BSD sockets API) allow for emulation of UDP connections. In this case connection latching can be supported using either model given above. We ignore, in this section, the fact that the connection latching model described in Section 2.3 can support per-datagram latching by extending its packet tagging interfaces to the application.

IPsec connection latch creation/release for UDP connections is triggered when:

- o An application creates a UDP "connection." This should cause the creation of an ESTABLISHED or BROKEN connection latch.
- o An application destroys a UDP "connection." This should cause the creation of an ESTABLISHED or BROKEN connection latch.

When a connection latch transitions to the BROKEN state and the application requested (or system policy dictates it) that the connection be broken, then UDP should inform the application, if there is a way to do so, or else it should wait, allowing the application-layer keepalive/timeout strategy, if any, to timeout the connection.

What constitutes an appropriate action in the face of administrative transitions of connection latches to the CLOSED state depends on whether the implementation's "connected" UDP sockets API provides a way for the socket to return an error indicating that it has been closed.

5.3. Connection Latching to IPsec for UDP with Datagram-Tagging APIs

Implementations based on either model of connection latching can provide applications with datagram-tagging APIs based on those described in Section 2.3. Implementations UDP with of the normative model of IPsec connection latching have to confirm, on output, that the application provided 5-tuple agrees with the application-provided connection latch; on input, UDP can derive the tag by searching for a connection latch matching incoming datagram's 5-tuple.

5.4. Connection Latching to IPsec for SCTP

SCTP [RFC4960] a connection-oriented protocol similar, in some ways, to TCP. The salient difference, with respect to connection latching, between SCTP and TCP is that SCTP allows each end-point to be identified by a set of IP addresses, though, like TCP, each end-point of an SCTP connection (or, rather, SCTP association) can only have one port number.

We can represent the multiplicity of SCTP association end-point addresses as a multiplicity of 5-tuples, each of which with its own a connection latch. Alternatively we can extend the connection latch object to support a multiplicity of addresses for each end-point. The first approach is used throughout this document, therefore we will assume that representation.

Of course, this approach results in N x M connection latches for any SCTP associations (where one end-point has N addresses and the other has M), whereas the alternative requires one connecton latch per-SCTP association (with N + M addresses). Implementors may choose either approach.

IPsec connection latch creation/release for SCTP connections is triggered when:

- An SCTP listener end-point is created (e.g., when the SCTP sockets listen() function is called on a socket). This should cause the creation of a LISTENER connection latch for each address of the listener.
- An SCTP INIT chunk is received on an IP address and port number for which there is a listener. This should cause the creation of one or more ESTABLISHED or BROKEN connection latches, one for each distinct 5-tuple given the client and server's addresses.
- An SCTP INIT chunk is sent (e.g., as the result of a call to the SCTP sockets connect() function). This should cause the creation of one or more ESTABLISHED or BROKEN connection latches.
- An SCTP ASCONF chunk [RFC5061] adding an end-point IP address is sent or received. This should cause the creation of one or more ESTABLISHED or BROKEN connection latches.

- Any state transition of an SCTP association to the CLOSED state will cause a corresponding transition for any associated connection latches to the CLOSED state as well.
- An SCTP ASCONF chunk [RFC5061] deleting an end-point IP address is sent or received. This should cause one or more associated connection latches to be CLOSED.

When a connection latch transitions to the BROKEN state and the application requested (or system policy dictates it) that the connection be broken, then SCTP should inform the application, if there is a way to do so, or else it should wait, allowing SCTP path/endpoint failure detection (and/or application-layer keepalive strategy) to timeout the connection. When a connection latch is administratively transitioned to the CLOSED state then SCTP should act as though an ABORT chunk has been received.

6. Security Considerations

6.1. Impact on IPsec

Connection latching effectively adds a mechanism for dealing with the existence, in the SAD, of multiple non-equivalent child SAs with overlapping traffic selectors. This mechanism consists of, at minimum, a local notification of transport protocols (and, through them, applications) of the existence of such a conflict which affects a transport layer's connections. Affected transports are also notified when the conflict is cleared. The transports must drop inbound packets, and must not send outbound packets for connections which are affected by a conflict. In this minimal form connection latching is a passive, local feature layered atop IPsec.

Connection latching achieves this by adding a new type of IPsec database, the Latch Database (LD), containing objects which represent a transport protocol's interest in protecting a given packet flow from such conflicts. The LD is managed in conjunction with updates to the SAD and the SPD, so that updates to either which conflict with established connection latches can be detected. For some IPsec implementations this may imply significant changes to their internals. However, two different models of connection laching are given, and we hope that most native IPsec implementators will find one model to be significantly simpler to implement than the other, and simple enough to implement.

This notion of conflicting SAs and how to deal with the situation does not modify the basic IPsec architecture -- the feature of IPsec which allows such conflicts to arise remains, and it is up to the transport protocols and applications to select whether and how to

respond to them.

There are, however, interesting corner cases in the normative model of connection latching that implementors must be aware of. The notes in Section 2.2.1 are particularly relevant.

6.2. Impact on IPsec of Optional Features

Section 3 describes optional features of connection latching where the key manager takes on a somewhat more active, though still local, role. There are two such features: optional protect/bypass, and preservation of "logical" SPD entries to allow latched connections to remain in the ESTABLISHED state in the face of adverse administrative SPD (but not SAD) changes. These two features interact with administrative interfaces to IPsec; administrators must be made aware of these features, and SHOULD be given a way to break ESTABLISHED connection latches. Also, given recent trends toward centralizing parts of IPsec policy, these two features can be said to have non-local effects where they prevent distributed policy changes from taking effect completely.

6.3. Security Considerations for Applications

Connection latching protects 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 may not be protected any more or less by use of IPsec connection latching than by use of IPsec alone without connection latching. 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.

Connection latching protects against TCP RST attacks. It does not help, however, if the original peer of a TCP connection is no longer available (e.g., if an attacker has been able to interrupt the network connection between the two peers).

6.4. Channel Binding and IPsec APIs

IPsec channels are a pre-requisite for channel binding [RFC5056] to IPsec. Connection latching provides such channels, but the channel bindings for IPsec channels (latched connections) are not specified herein -- that is a work in progress [I-D.williams-ipsec-channel-binding].

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].

7. IANA Considerations

There are not IANA considerations for this document.

8. Acknowledgements

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

9. References

9.1. 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.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[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.

[RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.

[RFC5061] Stewart, R., Xie, Q., Tuexen, M., Maruyama, S., and M. Kozuka, "Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration", RFC 5061, September 2007.

9.2. 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 abstract interface between applications and IPsec", draft-ietf-btns-abstract-api-01 (work in progress), February 2008.

[I-D.williams-ipsec-channel-binding]
Williams, N., "Channel Bindings for IPsec Using IKEv2 and Public Keys", draft-williams-ipsec-channel-binding-00 (work in progress), March 2008.

[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).