Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 14, 2009

M. Shore
D. McGrew
K. Biswas
Cisco Systems
July 13, 2008

Network-Layer Signaling: Transport Layer
draft-shore-nls-tl-06.txt

Status of this Memo

Copyright Notice

Abstract

The RSVP model for communicating requests to network devices along a datapath has proven useful for a variety of applications beyond what the protocol designers envisioned, and while the architectural model generalizes well the protocol itself has a number of features that limit its applicability to applications other than IntServ. Network Layer Signaling uses the RSVP on-path communication model and provides a lightweight transport layer for non-QoS signaling applications, such as discovery or diagnostics. It is based on a "two-layer" architecture that divides protocol function into transport and application. This document describes the transport protocol.

Table of Contents

1. Introduction

RSVP is based on a "path-coupled" signaling model, in which signaling messages between two endpoints follow a path that is tied to the data path between the same endpoints, and in which the signaling messages are intercepted and interpreted by RSVP-capable routers along the path. While RSVP was originally designed to support QoS signaling for Integrated Services [RFC1633], this model has proven to generalize to other problems extremely well. Some of these problems include topology discover, communicating with firewalls and NATs, discovery of IPSec tunnel endpoints, test applications, diagnostic triggers, and so on.

This document describes the core protocol for a generalized on-path request protocol that is being used today to carry topology discovery and other requests -- one that is not tied directly to IntServ or other QoS mechanisms and in which the protocol machinery itself is sufficiently generalized to be able to support a variety of applications (this protocol is referred to as "Network Layer Signaling", or "NLS"). What this means in practice is that there will be different signaling applications, all of which share a base NLS transport layer. This architecture is based on work done by Bob Braden and Bob Lindell, and described in [braden]. It is also similar to the concepts used in secsh, where authentication and connection protocols run on top of a secsh transport protocol (see [RFC4251] for details).

The protocol machinery was originally based somewhat on RSVP [RFC2205] without refresh overhead reduction extensions [RFC2961], but in the process of generalization has lost many of the features that define RSVP, such as necessary receiver-oriented reservations and processing requirements at each node.

NLS differs from RSVP in several important ways. One of the most significant of these is that the transport protocol described in this document (NLS-TL) does not itself trigger reservations in network nodes. Reservations will be installed and managed by NLS applications, however some NLS applcations may not carry reservation requests at all (discovery protocols, for example). Because of this NLS-TL does not support reservation styles (those would be also be attributes of an application). Another significant difference is that that reservations may be installed by a NLS application in either a forward (from the sender toward the receiver) or backward (from the receiver toward the sender) direction -- this is application-specific.

Other possibly significant differences include that NAT traversal support is integrated into the message transport, and that NLS allows

an application to install reservations for paths that are bidirectional and asymmetric.Multicast is not supported in this version of the protocol.

NLS shares some basic design features with NSIS [RFC4080], which is another path-coupled protocol. However, unlike NLS, the NSIS transport provides reliable delivery of request messages (in NLS this is left to the application rather than the transport layer), and NLS was not designed with QoS signaling support in mind. We leave the question of reliable delivery up to the NLS application. Not every signaling application requires reliable delivery and by moving reliable delivery into the application layer we are able to keep complexity in the transport layer to a minimum, providing only the services which tend to need to be available in all on-path signaling applications. It is also the case that by providing a light-weight, datagram-based protocol we are able to rely on IP layer routing and forwarding for delivery and do not require a separate routing process.

The NLS Transport Layer is being used by PacketCable and the ITU-T to carry topology discovery requests, in a protocol called "Control Point Discovery" (CPD).

1.1. Transport layer

This document describes the transport layer. The NLS transport layer is as simple as we could make it, supporting two basic functions: routing and NAT traversal. The sources of complexity in signaling protocols tend to be the signaling applications themselves. Those applications have varying performance and reliability requirements, and consequently we feel that application-specific functions belong in the application layer.

The NLS transport layer is also relatively stateless. By "stateless" we mean that the transport layer does not itself create or manipulate state in participating nodes. By "relatively" we take exception to the previous assertion, in that the transport layer provides facilities for route identification and route pinning. This is an optimization, albeit a significant one, which allows NLS to be used without a separate route discovery process. Another source of state is in the case of NATs, where an NLS-TL request may trigger the creation of a NAT table mapping. However, this latter case does not create NLS-TL maintenance state.

An application may wish to support summary refreshes or other performance enhancements; that type of function is application-specific and requires no support from the transport layer.

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. NLS-TL Messages

2.1. Message Processing Overview

Unlike RSVP, NLS-TL has only one fundamental message type, and directionality is significant to the NLS application only. Three new attributes, HOP-BY-HOP, BUILD-ROUTE, and BIDIRECTIONAL, have been added in support of greater flexibility in the NLS application. For example, some applications which already know network topology or which run a separate routing protocol may choose to route hop-by-hop in a forward direction. Conversely, a topology discovery protocol may choose to route end-to-end in the return direction. Both of these would be departures from the Path/Resv message handling specified in RSVP.

The BUILD-ROUTE flag has been added to allow route discovery to be overloaded on top of basic messaging, much like the RSVP Path message. If the BUILD-ROUTE flag is present, NLS nodes store routing information carried in incoming HOP objects. They also overwrite routing information into the HOP TLV in outgoing NLS messages.

The HOP-BY-HOP flag is used to instruct an NLS-TL node to address an outgoing message to the address stored in routing state associated with the message flow. Typically this will be state accumulated through the use of the BUILD-ROUTE flag but it may be manually configured, installed through an external routing or discovery process, etc.

The BIDIRECTIONAL flag may be used to indicate that the application for which this NLS-TL message carries a payload must be executed in each direction. It may be used in combination with the HOP-BY-HOP flag in some circumstances, but typically it will be used with the HOP-BY-HOP flag set to 0.

Even with these departures, the basic operation of the protocol may made be similar to RSVP with the appropriate use of the new attributes. For example, a message may be injected into a network by the sender towards a receiver, routed end-to-end with the receiver's address in the destination address in the IP header. If the BUILD-ROUTE bit is set in the NLS header, entities along the path the message traverses will intercept it, store path state, act on (or not) the application payload data, and forward the message towards its destination. In NLS-TL, "path state" refers specifically to the unicast IP address of the previous hop node along with locally-relevant path information (for example, interface identifier).

When the message arrives at the receiver (or its proxy), the receiver may generate another NLS message in response, this time back towards

the original sender. As with the message in the forward direction, this message may be routed either end-to-end or hop- by-hop, depending on the requirements of the application. In order to emulate an RSVP Resv message, the HOP-BY-HOP is set to 1 and the BUILD-ROUTE bit is set to 0.

BUILD-ROUTE and HOP-BY-HOP must not be set in the same NLS-TL message, and BUILD-ROUTE and TEARDOWN MUST NOT be set in the same NLS-TL message.

2.1.1. Congestion Considerations

Transmission, loss response, and resend timings are out-of-scope for this document. Different NLS applications will have different transmission timing and resend characteristics and will need to be specified in a manner appropriate to each application. For example, a discovery application will need to behave differently from an application which requests and maintains state in middleboxes.

However, each NLS application MUST specify how it will handle message loss and MUST specify a backoff mechanism in the case where messages are retransmitted as a response to message loss.

Loss response for stand-alone NAT traversal is described in Section 7.

2.2. NAT Traversal Support

NAT traversal poses a particular challenge to a layered protocol like NLS. If we assume the use of discrete, opaque applications, one of which is NAT, interactions between other applications that make use of addresses (for example, firewall rules or QoS filter specs) and the NAT application are complicated. Either every application will need to be able to peek into NAT payloads and identify which address mapping is the one they need, or NATs supporting NLS will need to be able to parse and write into every application payload type. Neither approach is particularly robust, reintroducing a type of stateful inspection and constraining how applications can be secured.

Because of the desire to be able to have a variety of NLS applications successfully interact with NATs and because of the constraints described above, in NLS NAT is supported in the transport layer rather than in a separate application. Each address that needs translation is tagged, put into a NAT_ADDRESS TLV, and passed to the appropriate application at each NLS node. Application identification is based on tag contents. Note that NLS supports Network Address Translation for IPv4 only.

2.3. NLS-TL Message Format

NLS messages consist of an NLS-TL header followed by optional TLV fields followed by an optional application payload.

2.3.1. The NLS-TL Message Header

All NLS-TL messages (and by implication, all NLS messages) start with an NLS header. The header is formatted as follows:

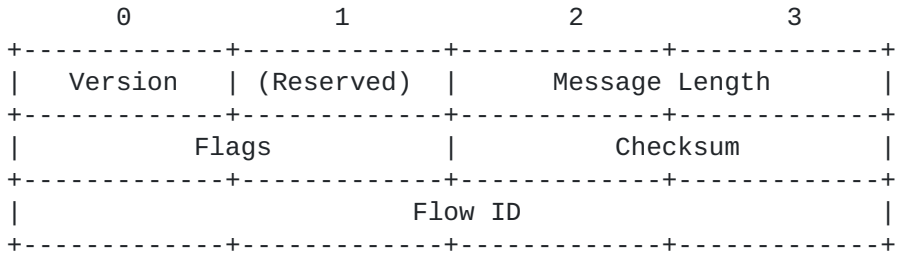


Figure 1

where the fields are as follows:

Version: 8 bits. The protocol version number; in this case 0x01.

Reserved: 8 bits. MUST be set to 0x0 when sending and ignored on receipt.

Message Length: 16 bits. The total number of octets in the message, including the NLS-TL header and complete payload.

Flags: 16 bits. Flag bits include

0x01 HOP-BY-HOP
0x02 BUILD-ROUTE
0X04 TEARDOWN
0x08 AX_CHALLENGE
0x10 AX_RESPONSE
0x20 BIDIRECTIONAL

Checksum: 16 bits. The one's complement of the one's complement sum of the entire message. The checksum field is set to zero for the purpose of computing the checksum. This may optionally be set to all zeros. If a message is received in which this field is all zeros, no checksum was sent.

Flow ID: 32 bits. This is a value which, combined with the source IP address of the message, provides unique identification of a flow of NLS-TL messages, which may be used for later reference for actions such as quick teardowns, status queries, etc. The mechanism used for generating the value is implementation-specific.

Throughout, we assume the use of 8-bit bytes, or octets.

2.3.2. NLS-TL TLVs

NLS-TL carries additional transport-layer information and requests as type-length-value fields, which are inserted after the header and before the application payload. The TLV format is as follows:

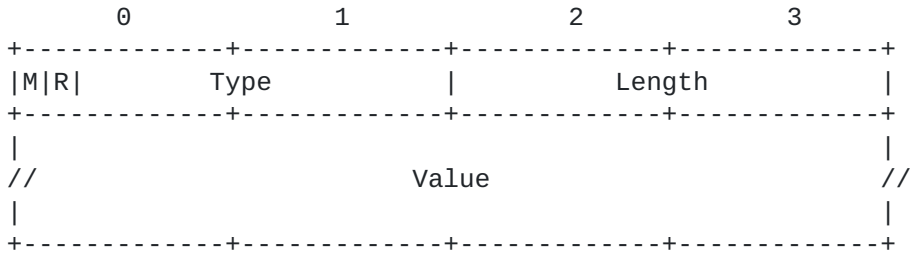


Figure 2

where the fields are as follows:

Mandatory: 1 bit. If this bit is set, this TLV MUST NOT be ignored silently, even if the recipient does not understand the type code. If the recipient does not understand the TLV, it MUST return an IPv4_ERROR_CODE or an IPv6_ERROR_CODE, as appropriate, with the error code set to 0x02 (Unrecognized TLV). If it is not set then the recipient MAY ignore the TLV.

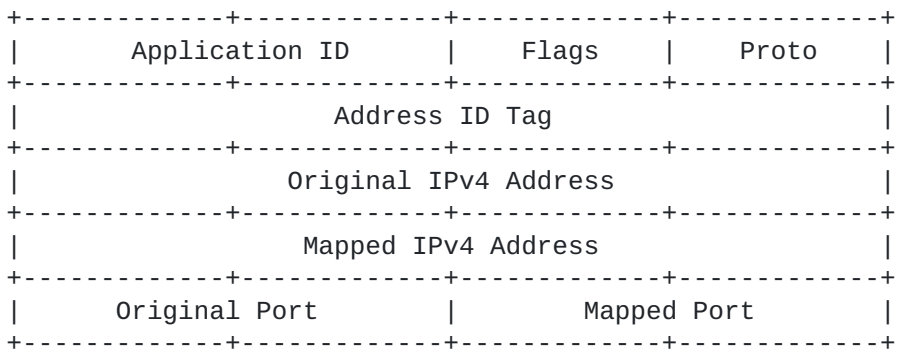
Reserved: 1 bit. This bit is reserved for future use. It MUST be set to 0 before sending and ignored on receipt.

Type: 14 bits. The type of information or request. Defined below.

Length: 16 bits. Total TLV length in octets, including the type type and length fields. It must always be at least 4 and be a multiple of 4.

Value: Variable length. At least 4 octets and a multiple of 4 octets). The TLV semantic content. The format of the Value field is determined by the value of the Type field

2.3.2.1. NAT_ADDRESS, TYPE=1



where the fields are as follows:

Application ID: 16 bits. This is the same as the value that's used for identifying application payloads. The Application ID field is set by the sender.

Flags: 16 bits. Flag bits include

0x01 = NO_TRANSLATE
0x02 = NO_REWRITE

NO_TRANSLATE indicates that a NAT device handling the packet should not create a NAT table entry for the original address. If the NO_TRANSLATE bit is set, the NAT does nothing.

NO_REWRITE indicates that when the reply message is being returned towards the sender, any NATs along the path MUST NOT overwrite the Mapped Address.

Proto: IP protocol for this translation (TCP, UDP, SCTP, etc.).

Address ID: 32 bits. An value that's unique within the set of Address IDs used with a particular Application ID; used to uniquely identify a particular address (i.e. provide a tag).

Original IPv4 Address: The original address for which a translation is being requested.

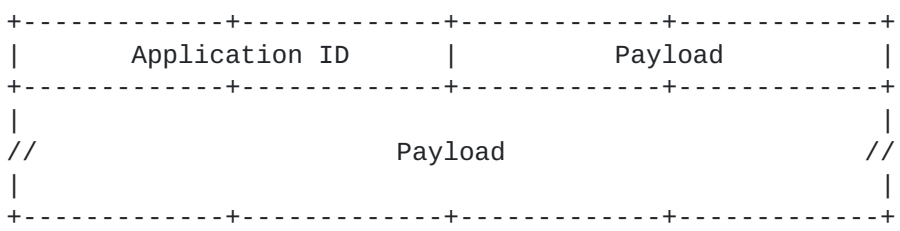
Mapped IPv4 Address: The address created by the NAT -- i.e. the "external" address.

Original Port: The original port for which a translation is being requested

Mapped Port: The port number created by the NAT for this mapping.

The mandatory bit in the TLV header MUST always be set to 1 for this TLV.

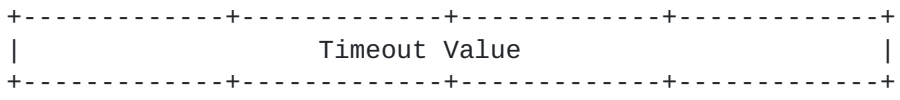
2.3.2.2. APPLICATION PAYLOAD, TYPE=2



The application payload TLV carries the NLS application data. It MUST follow any NAT TLVs. It consists of a 16-bit Application ID, which uniquely identifies the NLS application for which the TLV is intended, and the application payload itself. The application payload is transparent to the NLS Transport Layer.

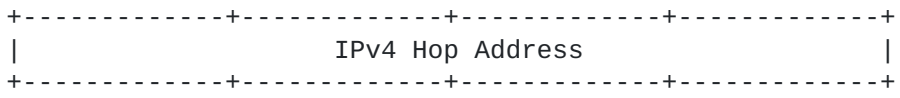
The APPLICATION_PAYLOAD TLV MUST be a multiple of 4 bytes in length.

2.3.2.3. TIMEOUT, TYPE=3



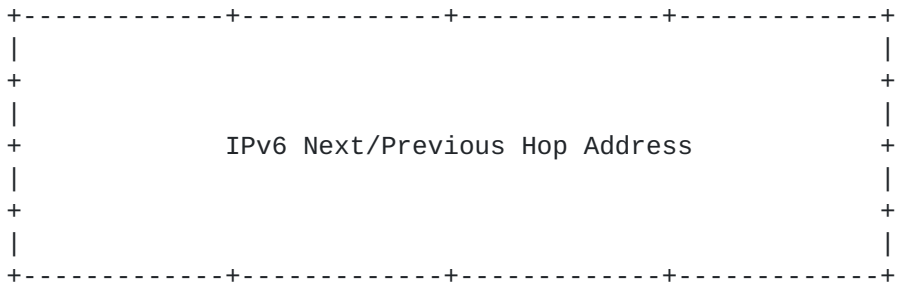
The TIMEOUT TLV carries the number of milliseconds for which state associated with a particular flow should be retained, with the expectation that the state will be deleted when the timeout expires. "State" in this case refers to routing state and to NAT state; NLS application state will be managed by its application.

2.3.2.4. IPV4_HOP, TYPE=4



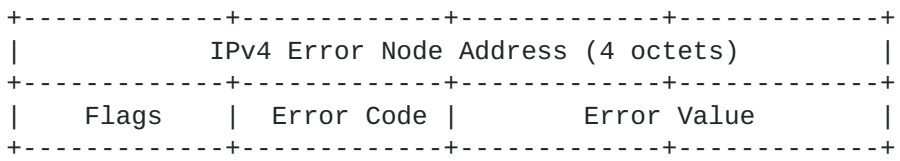
The IPv4_HOP TLV carries the IPv4 address of the interface through which the last NLS entity forwarded the message.

2.3.2.5. IPv6_HOP, TYPE=5



The IPv6_HOP TLV carries the IPv6 address of the interface through which the last NLS entity forwarded the message.

2.3.2.6. IPv4_ERROR_CODE, TYPE=6



The IPv4_ERROR_CODE TLV carries the address of a node at which an NLS-TL error occurred, along with an error code and error value. When no Error Value is defined, the Error Value field MUST be set to 0 by its sender and ignored by its receiver.

If the high-order bit of the Error Code is not set, the TLV carries an error message. If it is set, the TLV carries an informational message. Therefore Error Codes with values between 0 and 127 contain error messages and Error Codes with values between 128 and 255 contain informational messages.

IPv4 Error Node Address: 4 octets. The IPv4 address of the interface on the node that generated the error message.

Flags: 8 bits. None currently defined.

Error Code: 8 bits. The type of error or informational message, with values as follows:

Error Code = 0: No error

Error Code = 1: Bad parameters

Error Value = 1: HOP-BY-HOP and BUILD-ROUTE both present

Error Value = 2: BUILD-ROUTE present but no HOP TLV

Error Value = 3: HOP-BY-HOP present but no local stored routing state

Error Value = 4: Message length not a multiple of 4

Error Code = 2: Unrecognized TLV

Error Value = TLV number

Error Code = 3: Unrecognized application

Error Value = Application ID

Error Code = 4: Non-NLS NAT detected in path

Error Code = 5: Security error

Error Value = 1: AGID not found

Error Value = 2: Insufficient authorization

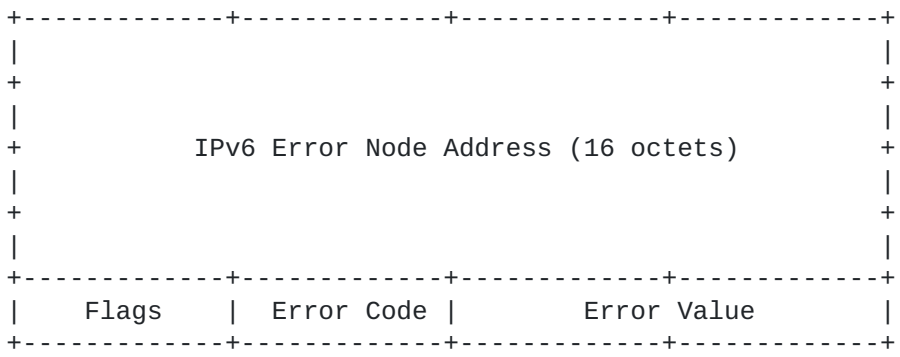
Error Value = 3: Request/reply mismatch

Error Value = 4: Authentication Failure

Error Code = 128: No message

Error Code = 129: Sending node has detected a route change

2.3.2.7. IPv6_ERROR_CODE, TYPE=7



The IPv6_ERROR_CODE TLV carries the address of a node at which an NLS-TL error occurred, along with an error code and error value.

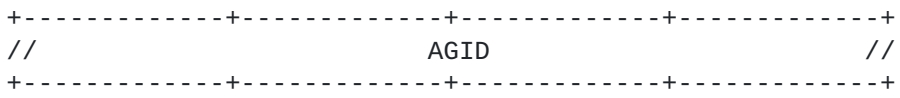
"IPv6 Error Node Address:" 16 octets. The IPv6 address of the interface on the node that generated the error message.

Flags: 8 bits. None currently defined.

The Error Code and Error value fields are the same as those used in the IPv4_ERROR_CODE.

2.3.2.8. AGID, TYPE=8

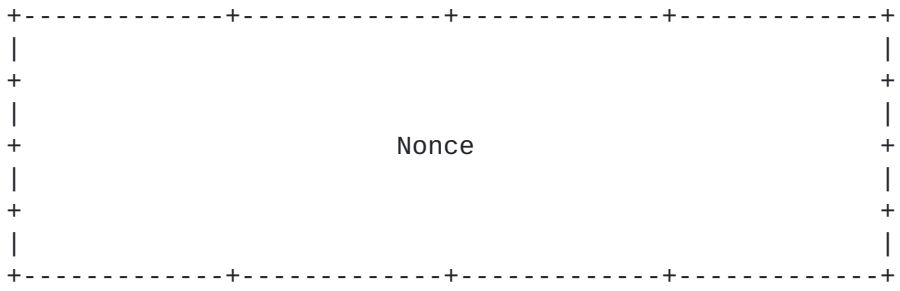
The AGID is the authentication group ID, used in the authentication dialogue to identify the group key.



2.3.2.9. A_CHALLENGE, TYPE=9

The A_CHALLENGE TLV is used to carry a 16-octet random nonce to be used as an authentication challenge. It MUST be generated using a strong random or pseudorandom source.

For a description of why we need A_CHALLENGE and B_CHALLENGE (as opposed to just a single CHALLENGE type), see Section 12.3.2.



2.3.2.10. A_RESPONSE, TYPE=10

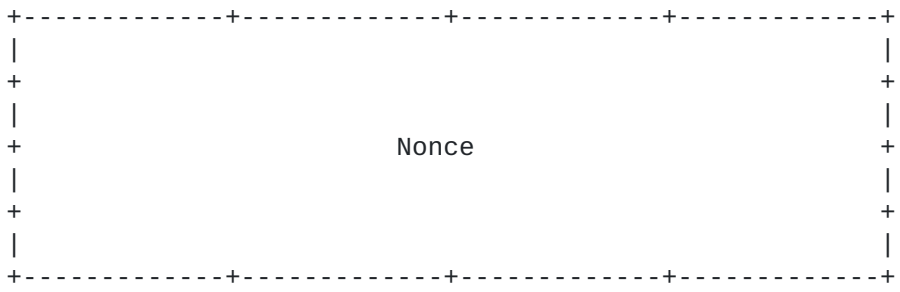
The A_RESPONSE TLV carries the response to the authentication challenge. It is a variable length TLV with the length dependent on the transform being used.

For a description of why we need A_RESPONSE and B_RESPONSE (as opposed to just a single RESPONSE type), see Section 12.3.2.

2.3.2.11. B_CHALLENGE, TYPE=11

The B_CHALLENGE TLV is used to carry a 16-octet random nonce to be used as an authentication challenge. It MUST be generated using a strong random or pseudorandom source.

For a description of why we need A_CHALLENGE and B_CHALLENGE (as opposed to just a single CHALLENGE type), see Section 12.3.2.



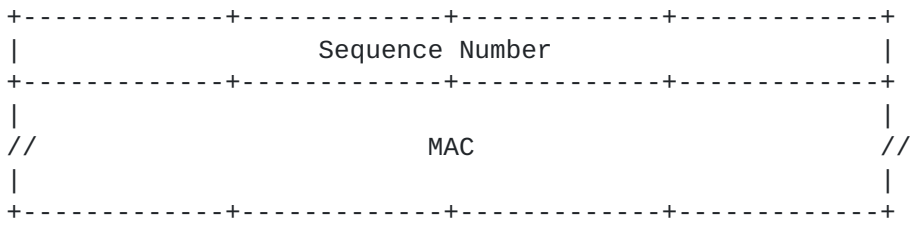
2.3.2.12. B_RESPONSE, TYPE=12

The B_RESPONSE TLV carries the response to the authentication challenge. It is a variable length TLV with the length dependent on the transform being used.

For a description of why we need A_RESPONSE and B_RESPONSE (as opposed to just a single RESPONSE type), see Section 12.3.2.

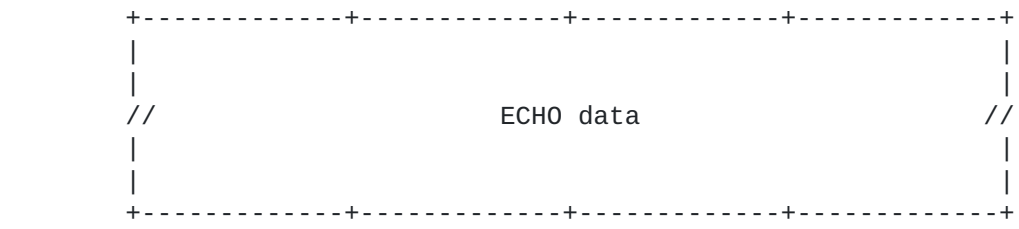
2.3.2.13. AUTHENTICATION, TYPE=13

The AUTHENTICATION TLV carries a cryptographic hash over the entire packet, as well as a 32-bit sequence number. In order to use this TLV, the peer must first have passed a challenge/response exchange to negotiate the approprite agid to use. It is a variable length TLV with the length of the MAC dependent on the transform being used (as determined by the agid). Details of computing the MAC are described in Section 4.6.



2.3.2.14. ECHO, TYPE=14

A device can include an ECHO element in the messages that it sends. A device receiving a message containing such a element must send the element back, verbatim, in the following response.



An ECHO TLV SHOULD only appear during an Authentication Exchange, and SHOULD NOT appear in any other message.

2.4. Cryptographic Datatypes

This section provides further detail on message formats for the authentication exchange.

An NLS-TL message MSG has the following format:

MSG :== HDR OPT* APP* SEC*

where HDR, OPT, APP, and SEC are as follows:

HDR is the NLS header

OPT is an NLS optional TLV

APP is the optional Application Object

SEC is an AGID, A_CHALLENGE, A_RESPONSE, B_CHALLENGE, B_RESPONSE, or AUTHENTICATION TLV's. These datatypes are defined below.

Note that though both OPT and APP are optional, one or the other MUST exist (or both together).

The security TLVs are always last in order to avoid data-formatting issues with the inputs to the message authentication codes, and to minimize the amount of data movement needed during the Authentication Exchange.

Authorization Group Identifier (AGID): The AGID TLV identifies a particular group key. The Value field carries an identifier; there is no defined format. The length of this field is variable, and MUST be a multiple of four octets. If it is generated at random, then it SHOULD be at least 16 octets.

A_CHALLENGE: The A_CHALLENGE contains a 16-octet random nonce. This TLV is put into a message whenever outbound authentication is desired. When this TLV is received, then the next message sent MUST contain either an A_RESPONSE TLV or an error message indicating that no authentication is possible.

B_CHALLENGE: The B_CHALLENGE contains a 16-octet random nonce. This TLV is put into a message whenever inbound authentication is desired. When this TLV is received, then the following message MUST contain either a B_RESPONSE TLV or an error message indicating that no authentication is possible.

A_RESPONSE: The A_RESPONSE TLV is sent in response to a message containing an A_CHALLENGE TLV. It contains a message authentication code (MAC) value computed over the complete NLS message containing the A_CHALLENGE, including the NLS header.

B_RESPONSE: The B_RESPONSE is sent in response to a message containing a B_CHALLENGE TLV. It contains a message authentication code (MAC) value computed over the complete NLS message containing the B_CHALLENGE, including the NLS header.

3. Sending NLS-TL Messages

When an endhost or its proxy wishes to initiate a NLS session, it creates an NLS-TL message. If the message is being sent end-to-end the destination address in the IP header is the address of the device interface that is expected to terminate the path along which signaling is expected to be sent (n.b. multicast is not supported in this version of the protocol). It may be a application peer host or terminal, or it may be a proxy. If the message is being sent hop-by-hop the destination address in the IP header is the address of the device interface that is the next hop along the path. That address will have been discovered either through a separate routing process or through RSVP-style soft-state messaging.

NLS-TL messages are UDP-encapsulated and sent on UDP port 7549. They MAY be sent with the router alert bit set in IPv4 headers or with the IPv6 router alert option [RFC2711], but it is not required. If the message is end-to-end and needs route discovery and pinning, the BUILD-ROUTE bit in the NLS-TL flags header MUST be set to 1 and the HOP-BY-HOP bit MUST be set to 0. If the message is being routed hop-by-hop, the HOP-BY-HOP bit MUST be set to 1 and the BUILT-ROUTE bit MUST be set to 0. (Note that there may be applications in which both the HOP-BY-HOP and the BUILD- ROUTE bit will be set to 0.)

If the NLS application wishes to support bidirectional reservations, the BIDIRECTIONAL flag must be set to 1, the BUILD-ROUTE flag should be set to 1, and the HOP-BY-HOP flag should be set to 0, at least in the initial message. If the application makes use of periodic refreshes it may optionally choose to route some number of them hop-by-hop along the discovered path before sending out another message to refresh the route state; that is an application design issue.

In this version of the protocol, each NLS message must fit in one datagram. An NLS-TL message originator should perform PMTU discovery in order to avoid exceeding path MTU size.

4. Messaging and state maintenance

Message handling and state maintenance are determined by the presence (or absence) of two flags in the NLS-TL header: the HOP-BY-HOP bit and the BUILD-ROUTE bit. They also involve, and are involved by, NAT processing.

4.1. BUILD-ROUTE

The BUILD-ROUTE bit in the flags field of the NLS-TL header allows NLS-TL to function as a discovery and routing protocol, much like the Path message described in RFC 2205.

If the BUILD-ROUTE flag is present in a NLS-TL message, upon receipt a NLS node MUST check for the presence of an IPv4_HOP or IPv6_HOP TLV in the NLS-TL payload. If one is not present, the message MUST be discarded and an error returned to the sender. If both are present, the message MUST be discarded and an error returned to the sender. Otherwise, if there is no installed soft state associated with the Flow ID, the node stores the HOP information, Flow ID, and other state information it chooses to retain, and forwards the message towards the address in the destination field of its IP header. If there is installed soft state associated with the Flow ID, the node compares the contents of the HOP field with the installed state. If they are identical nothing needs to be done; if they are different the HOP information in the node is overwritten with the information in the current message. This allows the protocol to be responsive to route changes, endpoint mobility, and so on.

A NLS node MAY send notification of a routing change back to the sender.

4.1.1. Route state deletion

Routing state accumulated through BUILDROUTE MAY be deleted in one of the following circumstances: 1) when the state is explicitly torn down in a message with the TEARDOWN bit set; 2) following the timeout period as described in Section 2.3.2.3, or 3) as a function of implementation-specific resource management.

4.2. HOP-BY-HOP

If the HOP-BY-HOP bit is set in the flags field of the NLS-TL header, a NLS node MUST forward the message to the address stored in associated local soft state. That is to say, the node MUST write the address in the local HOP information associated with the MESSAGE_IDFlow ID into the destination field in the IP header on the outbound message. This is like message processing in the Resv

message in RFC 2205.

The HOP information may have been acquired using a routing process based on HOP-BY-HOP processing, but it may have been acquired using an external routing mechanism. If there is no HOP information stored locally, the node MUST drop the message and return an error to the sender.

4.3. BIDIRECTIONAL

If the BIDIRECTIONAL flag is set, the receiver must send the answering message to the sender (that is to say, the destination address in the IP header must be set to the address of the sender) with the BUILD_ROUTE flag set and the HOP_BY_HOP flag set to 0. As with the message sent from the sender to the receiver, the HOP TLV contains information used to install routing state. If the nodes are already authenticated to one another (they were already traversed in the forward direction) it is unnecessary for the authentication dialogue to be performed again. If the nodes are not already authenticated to one another then the route is asymmetric and the authentication dialogue must be performed.

Note that the sender and receiver should retain knowledge that the session is bidirectional, as it may affect subsequent messaging and error processing.

Because a complete authentication dialogue may take place in each direction, with each node being authenticated to its adjacent node (i.e. the dialogue takes care of authenticating both A to B and B to A), this proposal neither changes the authentication dialogue nor should it undermine the security of the protocol.

4.4. Path Teardown Messages

Receipt of a NLS message with the TEARDOWN bit set indicates that matching path state must be deleted. Note that this is independent of directionality, and the teardown message may be sent in either direction. The applications which have reservations that were installed by a message containing a matching Flow ID must be notified, and they are responsible for managing (in this case, deleting) their own flow-related state. TEARDOWN and HOP-BY-HOP MUST NOT be set in the same message.

Unlike RFC 2205, if there is no matching path state the teardown message must be forwarded. There may be path state in support of an NLS application that is not running on every node, and the teardown message must not be lost.

4.5. Network Address Translation

If there is one or more NAT_ADDRESS TLVs present, an NLS-capable NAT must process each one that has does not have the NO_TRANSLATE bit set in the flags field. Processing takes place as follows:

- The originator (sender) of the message creates a NAT_ADDRESS TLV for each address/port/protocol tuple requiring NAT mappings. It also creates a random 32- bit tag, which is used to identify the address in application payloads and to tag the mapping in the NAT_ADDRESS TLV in the NLS-TL header. It also sets the TRANSLATE bit in the flags field and zeros the Mapped Address field.

- When an NLS-capable NAT receives a request, for each NAT_ADDRESS TLV in which the NO_TRANSLATE bit is not set and the Mapped Address is all nulls, it creates a NAT table mapping for the Original Address and Original Port and inserts the "external" address and port into the Mapped Address and Mapped Port fields.

- When an NLS-capable NAT receives a request, for each NAT_ADDRESS TLV in which the NO_TRANSLATE bit is not set and the Mapped Address is not nulls, it creates a NAT table mapping for the Mapped Address and Mapped port and overwrites those values with the new external addresses and ports.

- When an NLS-capable node receives a request, for reach NAT_ADDRESS TLV in which the Application ID matches an NLS application payload ID and the application is supported by the node, the TLV is passed to the application with the application payload, allowing the application module on the node to correlate and use the address based on the tag [and the Original Address?]

Note that this approach to NAT requires that participants be sensitive to directional issues in cases where ordering matters, such as the need to find the outermost NAT address. API support is required in order to turn the NO_TRANSLATE bit on and off as needed by a particular application.

Also note that in cases where the only function required is NAT table mapping requests, there may be no application payloads, or it may be desirable to create a rudimentary NAT NLS application that does nothing other than allow the receiver, or other nodes, to turn the NO_TRANSLATE bit on.

4.6. Authentication Exchange

NLS provides its own message authentication mechanism, based on a dialogue between adjacent nodes. We refer to this as the

"Authentication Exchange," or AX.

4.6.1. Authentication Exchange Messages

In the following, we consider only the Security TLVs, and we use REQUEST and REPLY to represent the body of the messages.

1. A -> B : HDR1, REQUEST, AGID*, A_CHALLENGE

2. B -> A : HDR2, REQUEST, AGID, B_CHALLENGE, A_RESPONSE

3. A -> B : HDR3, REQUEST, AGID, B_RESPONSE

/* at this point, B might forward the REQUEST onward */

4. B -> A : HDR4, REPLY, AUTHENTICATION

Note that the Flow ID in each message in the Authentication Exchange MUST be the Flow ID that appeared in the original request.

Note as well that the fields MUST be presented in the order specified, or the MAC calculations will fail.

Message 1 (outbound). When device A sends a message, it constructs the message as follows:

- It consults the policy associated with that interface to determine which AGID values should be included in that message. For each AGID in the policy that is associated with the Application ID in the message, it includes in that message an AGID TLV containing the AGID value.

- After the AGID TLVs have been included, an A_CHALLENGE TLV is constructed and included in the message.

- In the NLS-TL header for message 1, the AX_CHALLENGE flag must be set.

Message 1 (inbound). Device B receives (or intercepts) Message 1 and processes it as follows:

- The local policy associated with the interface on which the message arrived is checked to determine which AGIDs are associated with the Application ID in the message. If the AGID set in the message intersects with the locally derived AGID set, then one of the AGID values is chosen to be 'active'; this choice is arbitrary. Otherwise, the AX cannot be successfully completed, and an "AGID not found" error message SHOULD be returned.

Message 2 (outbound). Device B constructs Message 2 as follows:

- The NLS header is identical to that of Message 1, except that the AX_CHALLENGE and AX_RESPONSE flags are now set. The TLVs from Message 1 are copied verbatim into Message 2, in order, except for the AGID TLVs and the A_CHALLENGE TLV. A single AGID TLV containing the active AGID is appended to Message 2, followed by a B_CHALLENGE TLV and an A_RESPONSE TLV. The B_CHALLENGE TLV is constructed by generating its Nonce field uniformly at random. The A_RESPONSE TLV contains a message authentication code (MAC) value computed over the complete Message 1, also containing the A_CHALLENGE, the B_CHALLENGE, the A_RESPONSE (set to zero for the purpose of the MAC calculation) and including the NLS header, using the secret key associated with the AGID. Device B then sends Message 2 to Device A.

- In the NLS-TL header in Message 2, the AX_CHALLENGE and AX_RESPONSE flags must be set

- If the optional ECHO TLV is used, it must be placed after the AGID (i.e. between the AGID and the A_CHALLENGE)

For the purpose of the MAC calculation for A_RESPONSE, the "entire NLS message" is:

HDR1||REQUEST||AGID||A_CHALLENGE||A_RESPONSE||B_CHALLENGE

Message 2 (inbound). Device A processes Message 2 by performing the following checks:

- Verifying that the AGID in the message is associated with the Application ID in the NLS message. If it is not, then the Authentication Exchange cannot be successfully completed, an error message of "Insufficient authorization" SHOULD be returned, and the connection MUST be abandoned.

- Verifying that the TLVs other than the security TLVs in Message 2 match the non-security TLVs in Message 1. The two messages should be bitwise identical, besides the security TLVs (and the transport headers below the NLS header). If the messages do not match, then the Authentication Exchange cannot be successfully completed, an error message of "Request/reply mismatch" SHOULD be returned, and the connection MUST be abandoned.

- If the other checks pass, then Device A computes its own value of the A_RESPONSE TLV, using as input the key associated with the

AGID in the message, and the locally cached copy of Message 2. Note that it may be necessary to make a temporary copy of the value of the A_RESPONSE MAC field before setting that field to zero, in order to compare the locally computed value to the received value.

- If the locally constructed A_RESPONSE does not match the A_RESPONSE in Message 2, then the Authentication Exchange cannot be successfully completed, an error message of "Authentication failure" SHOULD be returned, and the connection MUST be abandoned.

If all of those those steps are passed, then Message 3 is computed as described below.

- Message 3 (outbound): Device A constructs Message 3 as follows. The NLS header is identical to that of Message 2, except that the AX_RESPONSE flag is set, and the AX_CHALLENGE flag is not set. The TLVs from Message 2 are copied verbatim into Message 3, in order, except for the A_RESPONSE TLV. An A_CHALLENGE and B_RESPONSE TLV are appended to Message 3. The B_RESPONSE TLV is constructed by computing the MAC over the entire NLS message from the header up to and including the B_RESPONSE TLV (with the MAC field set to zero), using the secret key associated with the AGID. Device A then sends Message 3 to Device B.

- In the Message 3 NLS-TL header, the AX_RESPONSE flag must be set

- If the optional ECHO TLV is used, it MUST follow the AGID (i.e. between the AGID and the B_RESPONSE).

For the purpose of the MAC calculation for B_RESPONSE, the "entire NLS message" is:

HDR2||REQUEST||AGID||B_RESPONSE||A_CHALLENGE||B_CHALLENGE

Message 3 (inbound): Device B processes Message 3 by performing the following checks:

- Verifying that the AGID in the message is associated with the Application ID in the NLS message. If it is not, then the Authentication Exchange cannot be successfully completed, an error message of "Insufficient authorization" SHOULD be returned, and the connection MUST be abandoned.

- Computing its own value of B_RESPONSE, by computing the MAC over the entire NLS message from the header up to and including the

B_RESPONSE TLV (with the MAC field set to zero), using the secret key associated with the AGID. If the locally constructed B_RESPONSE does not match the one received in Message 3, then the message is rejected, and an error message of "Authentication failure" SHOULD be returned. Note that it may be necessary to make a temporary copy of the value of the B_RESPONSE MAC field before setting that field to zero, in order to compare the locally computed value to the received value.

After an authentication exchange has completed sucessfully and a single AGID has been negotiated, the nonces sent to and received from the peer MUST both be saved for use with the AUTHENTICATION TLV. Also, the sequence number associated with the nonces MUST be set to 0 immediately after finishing the exchange successfully, and before using an AUTHENTICATION TLV. Should any previous state exist (i.e. previous nonces and sequence numbers), these MUST be replaced by the new nonces (and the sequence numbers reset to 0).

After these checks pass, then the body of the NLS message, with the B_RESPONSE TLV removed, is proccessed by the NLS application, that is, it is processed in the manner determined by the Application ID.

4.6.2. Authentication TLV calculation

The AUTHENTICATION TLV is calculated over the entire packet (as described in the next paragraph) as well as the nonce from the last challenge received from (or sent to, in the case of the receiver) the peer (from either A_CHALLENGE or B_CHALLENGE). The nonce is associated with a sequence number, which helps guard against replay attacks. The AUTHENTICATION TLV MUST be at the end of the TLV stream.

The appropriate MAC algorithm to be used is negotiated in a previous Challenge/Response exchange, where AGID TLV's were exchanged and one single AGID was agreed upon.

The sender:

- Add an empty AUTHENTICATION TLV to the end of the TLV stream (i.e. with the HMAC field set to all 0's).
- Find the last nonce received from the peer in either an A_CHALLENGE or B_CHALLENGE.
- Increment the sequence number associated with the nonce by one, and write it into the 'sequence number' field of the AUTHENTICATION TLV

- Calculate the HMAC from the concatenation of the entire packet (with header and the incomplete AUTHENTICATION TLV) and the nonce. Write the resulting HMAC value into the HMAC field of the AUTHENTICATION TLV.

- Send the packet.

The receiver:

- Copy the HMAC field from the AUTHENTICATION TLV into local storage, and overwrite the HMAC value in the packet with all 0's. Find the last nonce sent to the peer in either an A_CHALLENGE or B_CHALLENGE

- Calculate the HMAC from the concatenation of the entire packet (with header and the incomplete AUTHENTICATION TLV) and the nonce.

- Compare the calculated value to the value copied out int he first step.

- If the values match, check to see if the sequence number falls into the range of valid sequence number (as determined by a sliding window), and if so, the sliding window is updated.

- If the values do NOT match, the packet MUST be discarded and an error message SHOULD be returned to the sender (rate-limited to prevent DoS attacks).

The sliding window SHALL be done according to [RFC4303] Appendix A2.

4.6.3. Security state transition table

The security state transitions are as follows:

State name	Event	Transition next state
Closed	Send unprotected message	Closed
Closed	Send Message 1	Waiting for Message 2
Closed	Accept Message 1, send Message 2	Waiting for Message 3
Waiting for Message 2	Timeout expired	Closed
Waiting for Message 2	Reject invalid message	Closed
Waiting for Message 2	Accept Message 2	Secure connection established
Waiting for Message 3	Timeout expired	Closed
Waiting for Message 3	Reject invalid message	Closed
Waiting for Message 3	Accept Message 3	Secure connection established
Secure connection established	Send authenticated message	Secure connection established

5. Application Interface

Application payloads are encapsulated within NLS-TL TLVs, and MUST follow any NAT TLVs.

The Application Payload TLV carries includes the Application ID field, which is used to vector the requests off to the correct application on the router upon receipt. It is also used to identify NAT_ADDRESS TLVs to be passed to the application. In a nutshell, if the Application ID in a NAT_ADDRESS TLV matches the Application ID in an Application TLV, the NAT_ADDRESS TLV must be passed to the application along with the application payload.

The Length field carries the total application payload length, excluding the header, in octets. The length must be at least 4 and be a multiple of 4. It may be necessary for an application to pad its payload to accomplish that.

Note that there is no identifier in the TLV other than the Application ID. If there is a need for an application-specific identifer for reservations or other applications requiring retained state, those must be added to the application payload.

6. NAT Interactions

NLS uses IP addresses for routing, both end-to-end and hop-by-hop. Given the applications which NLS-TL will be transporting, it is highly likely that those applications will be using payload-embedded addresses and there will be some interactions. The use of a NAT application together with other applications can mitigate this, but there will be problems transiting non-NLS-capable NATs.

When an NLS entity receives an TL message travelling in the forward direction, it writes the address in the IPv4_HOP or IPv6_HOP, as appropriate, from the packet into local per-session state and replaces the HOP data in the message with the address of the outgoing interface. When the entity is a NAT, it will write the translated-to address. Note that while it is usually the case that payload integrity protection breaks in the presence of NATs if embedded addresses are being rewritten, this is not substantially different from the rewriting of the HOP field which occurs within NLS anyway.

However, if an NLS message crosses a non-NLS-capable NAT, several problems may occur. The first is that if the message is being dropped in a raw IP packet, the NAT may simply drop the packet because it doesn't know how to treat it. Another is that the address in the HOP field will be incorrect. NLS and the applications it carries cannot be expected to function properly across non-participating NATs. Discovery of a non-NLS-capable NAT is described in Section 8.

7. Using NLS-TL as a stand-alone NAT traversal protocol

Using the NLS Transport Layer as a stand-alone NAT traversal protocol is straightforward -- simply use the TL without application payloads, but set the NO_REWRITE flag in the NAT_ADDRESS TLV to 1. This provides two functions: 1) installation of new NAT table mappings, and 2) allowing the sender to learn what the "external" mappings are. The Application ID field in the NAT_ADDRESS TLV must be set to 0.

The TL header flags in the forward direction must be

HOP-BY-HOP = 0

BUILD-ROUTE = 1

TEARDOWN = 0

The TL header flags in the reverse direction (i.e. in the response message) must be

HOP-BY-HOP = 1

BUILD-ROUTE = 0

TEARDOWN = 0

The NAT table mappings are kept fresh through the retransmission of the request every refresh period. The refresh messages are identical to the original request message.

If a response message is not received, the retransmission and backoff procedures described in Section 6 of [RFC2961] MUST be used.

When the NAT table mappings are no longer required, the sender must send a teardown message containing the Flow ID of the installed mappings and with the TL flags set to

HOP-BY-HOP = 0

BUILD-ROUTE = 0

TEARDOWN = 1

An acknowledgement response message is not required. If there has been no refresh message received prior to the expiration of the timeout period, the NAT table mappings must be deleted when the timeout period ends.

8. Discovery of non-NLS NATs, and recovery

This section describes a method of discovering non-NLS NATs in the path, and a recovery-mechanism if one is discovered.

When there are non-NLS-capable NATs in the path, they will only be able to process or modify the IP/UDP header of the NLS-TL message and will not be able to understand or modify the NLS-TL message itself (including the NAT_ADDRESS_TLV inside).

If there are non-NLS NATs in the path the sender needs to be made aware of this, and it should be able to fall back to processing without NLS, using any other mechanisms that may be available. Also, the NLS_ NATs in the path which have allocated the NAT mappings based on NLS NAT_ADDRESS_TLV processing, need to be able to release these mappings.

The following algorithm can be applied for non-NLS NAT detection by NLS nodes :

```
if (NAT_TL NAT_ADDRESS_TLV's mapped_addr == 0) {
    This NLS_TL NAT is first NLS_TL NAT in path
    if (NLS_TL packet's source IP address != NAT_ADDRESS_TLV's
    original_address) {
        This NLS_TL NAT is not the first in the path, and
        some non-NLS_TL NAT has touched this packet;
        send NLS_TL error message back to the sender
        with NLS_TL error-code = 4 (non-nls-nat in path)
    } else {
        This NLS_TL NAT is the first in the path, and no non-
        NLS_TL NAT has touched this packet;
        proceed with NLS_TL processing.
    }
} else {
    This NLS_TL NAT is not the first NLS_TL NAT in path.
    if (NLS_TL packet's source IP address != NAT_ADDRESS_TLV's
    mapped_address) {
        Some non-NLS_TL NAT has touched this packet, send
        NLS_TL error message back to the sender with NLS_TL
        error-code = 4 (non-nls-nat in path)
    } else {
        No non-NLS_TL NAT has touched this packet; proceed
        with regular NLS_TL processing.
    }
}
```

The NLS_TL error message will be relayed back to the sender. Intermediate NLS nodes should not be processing the NLS error message, but let this NLS packet be routed back to the sender.

Once the sender sees an NLS_TL error-message with Error-Code = 4 (non-nls-nat in path), it should resend the same NLS_TL message as earlier with the NAT_ADDRESS_TLV's Original IPv4 Address/Port/ Protocol as earlier and the Mapped IPv4 Address/Port as NULL, but should set the TEARDOWN flag in the NLS-TL header.

The intermediate NLS NATs in the path, upon seeing an NLS_TL message with the TEARDOWN bit set, should delete its local NAT mapping corresponding to the Flow ID and send the message on towards the receiver, traversing other NLS-capable NATs along the path which will also process the TEARDOWN message.

9. Endpoint Processing

This section describes the procedures used in the endpoints (that is to say, the sender and the receiver) for processing NLS packets. Note that these are the endpoints for the purposes of describing an end-to-end NLS path; they may actually be network entities or proxies.

9.1. Sending

When a host or its proxy wishes to send an NLS application request, it puts together the application payload and encapsulates it in a transport layer packet.

If the application needs to request NAT service because of its use of addresses for reservations, etc., it must create a random 32-bit tag for use as an address token in the application payload, and it must create a NAT_ADDRESS TLV in which it inserts the address and port for which it is requesting NAT service, as well as the 32-bit tag.

For example, in a hypothetical QoS application that needed NAT services for the address 192.0.2.110, TCP port 6603 in the flow description, it would generate the random tag 0x24924924, use that in the application payload instead of an address, and create a NAT_ADDRESS TLV with the following values:

Application ID = QoS

Flags = TRANSLATE

Proto = TCP

Address ID = 0x24924924

Original IPv4 Address = 192.0.2.110

Original Port = 6603

The endpoint also needs to set the flags that determine how path establishment and routing are to be handled on intermediate nodes. In some cases the application requires no stored state in NLS nodes or it simply requires a single NLS pass. Examples of this kind of application include topology discovery, tunnel endpoint discovery, or diagnostic triggers. In this case, in the NLS-TL header both the HOP-BY-HOP flag and the BUILD-ROUTE flag are set to 0.

If an application is establishing per-node state and wants the NLS transport layer to establish and pin NLS routing for it, as might be

the case with a QoS application or a firewall pinholing application, the sending endpoint must set the BUILD-ROUTE flag to 1 and the HOP-BY-HOP flag to 0.

The endhost then UDP encapsulates the NLS-TL packet, and transmits it on UDP port 7549.

9.2. Receiving

An NLS node "knows" that it's an endpoint or proxy when the following conditions are satisfied:

```
if (IP destination address == my address)  {
    if (HOP_BY_HOP)
        if (next hop data available)
            forward it on;
        else
            it's mine;
}
```

When an endpoint receives a packet and identifies it as terminating there, it demultiplexes the payload and passes the payload and associated NAT_ADDRESS data to the appropriate application.

If an application in the payload is not supported by the endpoint, the endpoint must return a message to the sender with an ERROR_CODE TLV with the error value set to 3 (Unrecognized application).

10. Intermediate node processing

The processing of NLS-TL packets at intermediate nodes is substantially the same as processing at endpoints. Upon the arrival of a request, the node demultiplexes the packet contents and vectors the application payloads off to their respective applications.

One major difference from endpoint processing is the handling of NAT requests by NAT intermediate nodes. When an NLS-capable NAT receives an NLS request, it checks for the presence of NAT_ADDRESS TLVs. For each NAT_TLV, it executes the process described in Section 4.5.

For state maintenance and forwarding, the node must follow the processes described in Section 4.1, Section 4.2, and Section 4.4.

11. Using NLS-TL to support bidirectional reservations

When an application that uses NLS-TL to transport reservation requests (for example, QoS reservations or firewall pinholes) and it wishes to make the request for a bidirectional data stream, the reservations should be made when the message is received in the "forward" direction. Note that this is a significant departure from the model used in RSVP and assumed in previous versions of NLS-TL. The reason for this should be apparent -- if the route between the sender and receiver is asymmetric, it is possible that a device traversed by a PATH message may not be traversed by a RESV message, and vice-versa.

It may be desirable to have different characteristics for the reservation in one direction than for the other. In this case the NLS application designer should make provision for identifying reservation specifications to be used in each direction.

It should also not be assumed, as is done in RSVP, that error messages will traverse all affected nodes unless care is taken by the sender, or the "owner" of the reservation, to ensure that error messages are propagated correctly. So, for example, if a reservation fails at a particular node, it may not be sufficient to return the error message towards the sender.

An application that manages reservations may wish to refresh application state more frequently than it wishes to refresh route state. In that case it should send the message with the BIDIRECTIONAL and HOP_BY_HOP flags set, and the BUILD_ROUTE flag set to 0.

12. Security Considerations

12.1. Overview

This section describes a method for providing cryptographic authentication to the Network Layer Signaling (NLS) transport layer protocol. The method incorporates a peer discovery mechanism. Importantly, there is no provision for confidentiality. This fact simplifies the protocol, and removes the need for export control on products implementing it. NLS applications which require confidentiality may provide it themselves.

This mechanism provides both entity and message authentication along a single hop. In other words, the device on each end of the hop is assured that the identity of the other device, and the content of the message from that device, are correct. These security services are provided only on a hop-by-hop basis. That is, there are no cryptogrpahic services provided across multiple hops, and each hop can independently use or not use authentication. In the following, we restrict our discussion to a single hop along an NLS path.

In order to support authentication, we introduce an optional two-message exchange into NLS called the Authentication Exchange, or AX. This exchange is needed in order to carry the challenge-response information, and is described in detail in Section 4.6.

12.2. Security Model

Authenticated NLS-TL provides both authorization and entity authentication using a group model. Authorizations correspond to particular applications. An Authorization Group (AG) is a set of network interfaces that share the following information:

- a list of NLS Application IDs; these correspond to applications which the group is authorized to use,
- a group authentication key,
- a Message Authentication Code (MAC) algorithm type

Note that AGs are associated with interfaces and not devices since in many situations there are different trust levels associated with different interfaces.

For each device implementing Authenticated NLS-TL, each interface is associated with a list of Application IDs, each of which is associated with:

- a list of AGIDs that authorize the corresponding application, or
- the symbol ALLOW, which indicates that the application has been explictly allowed on the associated interface, or
- the symbol DROP, which indicates that the application has been explicitly disallowed on the associated interface.

This model provides authorization at the NLS-TL layer. For example, it is impossible to authorize VoIP traversal of a firewall while still disallowing telnet across the firewall. The model can be expanded to accomodate finer grained authorizations, but this issue is not considered further in this draft. Sensitive applications, such as firewall pinholing, as well as applications requiring finer-grained authorizations, must provide their own authentication and authorization.

12.3. Cryptography

Authenticated NLS-TL uses a single cryptographic function: a pseudorandom function that accepts arbitrary-length inputs and produces fixed-length outputs. This function is used as a message authentication code (MAC).

The default MAC algorithm is HMAC SHA1, with a length truncated to 96 bits. No other message authentication code is defined. Other MACs MAY be implemented. Each key used in NLS is associated with a single MAC algorithm; thus crypto algorithm agility is supported by the same protocol mechanisms that support key agility. In particular, an NLS device can determine the MAC algorithm used by referencing the Value field of the Authorization Group ID, or AGID, (defined in Section 2.3.2.8).

12.3.1. Keys

Authenticated NLS-TL uses group keys, in order to reduce the amount of protocol state and to mitigate the peer-discovery problem.

Keys may be acquired or provisioned one of three ways: 1) through manual provisioning, 2) by being fetched from a central data store, or 3) through an automated key management system.

Implementations MUST provide a way to set and delete keys manually. However, they SHOULD also provide an automated group key management system such as GDOI [RFC3547], so that efficient revocation is possible.

When using an automated key management system, the AGID MAY be used

to differentiate key versions as well as authorization groups, to support rekeying. It can be expected to change over the lifetime of a message flow, while when using manually-provisioned keys the AGID may be expected to remain static.

12.3.2. Reflection Attacks

NLS is designed to resist reflection attacks. That family of attacks works against poorly designed mutual authentication systems by tricking one party into providing the response for its own challenge. In order to resist reflection attacks, distinct TLV types are defined for the first and second challenges, the A_CHALLENGE and B_CHALLENGE. This fact ensures that the two invocations of the MAC during a single challenge/response exchange will necessarily have different inputs, thus thwarting reflection attacks.

13. IANA Considerations

There are several parameters for which NLS-TL will need registry services. These include

- a registry for NLS Application IDs (NLS Application Identifiers) and for
- NLS-TL TLV identifiers (NLS TLVs).
- NLS-TL Error codes and error values

Initial values are given below. Future assignments are to be made through expert review.

NLS-TL also uses UDP port number 7549.

13.1. NLS Application Identifiers

NAME	VALUE	DEFINITION
Network Address Translation	0	NAT
Control Point Discovery	1	PacketCable CDP
Firewall Traversal	2	

Note that while there is no application payload for NAT, we define an application ID to support NLS-TL authentication when using an automated key management system. The details of the interface to a key management system are out of the scope of this document.

13.2. NLS TLVs

NAME	VALUE	DEFINITION
NAT_ADDRESS	1	See Section 2.3.2.1
APPLICATION_PAYLOAD	2	See Section 2.3.2.2
TIMEOUT	3	See Section 2.3.2.3
IPV4_HOP	4	See Section 2.3.2.4
IPV6_HOP	5	See Section 2.3.2.5
IPV4_ERROR_CODE	6	See Section 2.3.2.6
IPV6_ERROR_CODE	7	See Section 2.3.2.7
AGID	8	See Section 2.3.2.8
A_CHALLENGE	9	See Section 2.3.2.9
A_RESPONSE	10	See Section 2.3.2.10
B_CHALLENGE	11	See Section 2.3.2.11
B_RESPONSE	12	See Section 2.3.2.12
AUTHENTICATION	13	See Section 2.3.2.13
ECHO	14	See Section 2.3.2.14

13.3. Error codes and values

NAME	VALUE	DEFINITION
Error Code	0	No error
Error Code	1	Bad parameters
Error Value	1	HOP-BY-HOP and BUILD-ROUTE both present
Error Value	2	BUILD-ROUTE present but no HOP TLV
Error Value	3	HOP-BY-HOP present but no local stored routing state
Error Value	4	Message length not a multiple of 4
Error Code	2	Unrecognized TLV
Error Code	3	Unrecognized application
Error Code	4	Non-NLS NAT detected in path
Error Code	5	Security error
Error Value	1	AGID not found
Error Value	2	Insufficient authorization
Error Value	3	Request/reply mismatch
Error Value	4	Authentication Failure
Error Code	128	No message
Error Code	129	Sending node has detected a route change

14. References

14.1. Normative References

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

[RFC2205] Braden, B., Zhang, L., Berson, S., Herzog, S., and S. Jamin, "Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification", RFC 2205, September 1997.

[RFC2961] Berger, L., Gan, D., Swallow, G., Pan, P., Tommasi, F., and S. Molendini, "RSVP Refresh Overhead Reduction Extensions", RFC 2961, April 2001.

[RFC3547] Baugher, M., Weis, B., Hardjono, T., and H. Harney, "The Group Domain of Interpretation", RFC 3547, July 2003.

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, December 2005.

14.2. Informative References

[RFC1633] Braden, B., Clark, D., and S. Shenker, "Integrated Services in the Internet Architecture: an Overview", RFC 1633, June 1994.

[RFC2711] Partridge, C. and A. Jackson, "IPv6 Router Alert Option", RFC 2711, October 1999.

[RFC4080] Hancock, R., Karagiannis, G., Loughney, J., and S. Van den Bosch, "Next Steps in Signaling (NSIS): Framework", RFC 4080, June 2005.

[RFC4251] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture", RFC 4251, January 2006.

[braden] Braden, R. and R. Lindell, "A Two-Level Architecture for Internet Signaling", draft-braden-2level-signaling-01.txt (work in progress), November 2002.

Appendix A. Acknowledgements

The authors would like to express their gratitude to Jan Vilhuber, Senthil Sivakumar, Bill Foster, and Dan Wing for their careful review and feedback. Special thanks to Jan for his text on challenge/response calculations, and to Rajesh Karnik and Lisa Fang for their help in clarifying the text describing the authentication exchange. Brian Weis and Sheela Rowles provided invaluable discussion of automated group key management.

Authors' Addresses

Melinda Shore
Cisco Systems
809 Hayts Road
Ithaca, New York 14850
USA

Email: mshore@cisco.com

David A. McGrew
Cisco Systems
510 McCarthy Blvd
Milpitas, California 95035
USA

Email: mcgrew@cisco.com

Kaushik Biswas
Cisco Systems
510 McCarthy Blvd
Milpitas, California 95035
USA

Email: kbiswas@cisco.com

Full Copyright Statement

Intellectual Property

Acknowledgment

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