ABONE-DRAFT
Expiration: April 2000
File: draft-nodeos-security-00.txt

Bob Lindell
USC/ISI

Active Networks Protocol Specification for Hop-By-Hop Message Authentication and Integrity

Abstract

This document proposes a protocol specification of the Hop-By-Hop Message Authentication and Integrity described in the ABone Network Security Architecture [1]. It defines a new ANEP option to support this mechanism and provides guidance to implementing the associated packet processing and key management functions.

1. Introduction

Ensuring the integrity of active networks activities requires the ability to protect message traffic against corruption and spoofing. This document defines a mechanism to protect ANEP packets using hop-by-hop checks for message authentication and integrity. The proposed scheme transmits an authenticating digest of the message, computed using a secret Authentication Key and a keyed-hash algorithm. This scheme provides protection against forgery or message modification. The INTEGRITY option of each ANEP packet includes a one-time-use sequence number. This allows the message receiver to identify playbacks and hence to thwart replay attacks. The proposed mechanism does not afford confidentiality since messages stay in the clear. This decision not to include confidentiality was deliberate to avoid export restriction issues.

The message replay prevention algorithm is quite simple. The sender generates messages with monotonically increasing sequence numbers. In turn, the receiver only accepts messages that have a larger sequence number than the previous message. To start this process, a receiver handshakes with the sender to get an initial sequence number. This memo discusses ways to relax the strictness of the in-order delivery of messages as well as techniques to generate monotonically increasing sequence numbers that are robust across sender failures and restarts.

The proposed mechanism is independent of a specific cryptographic algorithm, but the document describes the use of Keyed-Hashing for Message Authentication using HMAC-

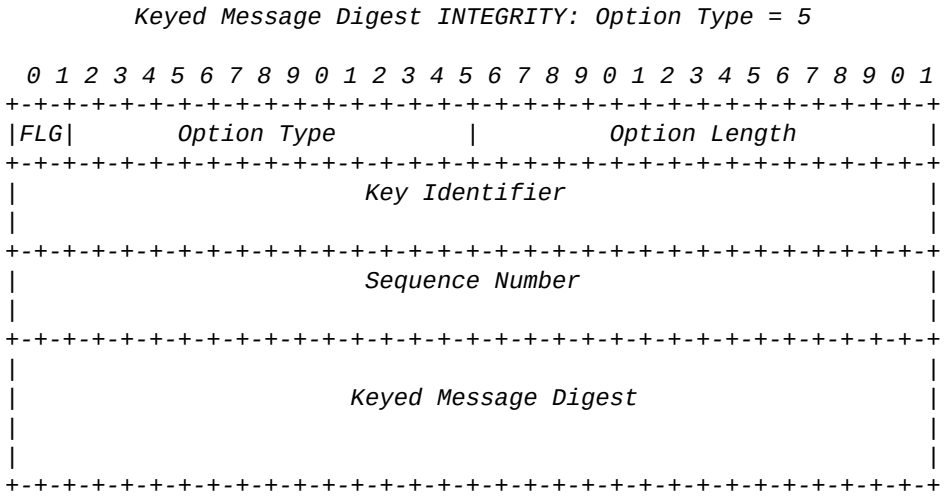
MD5 [2]. It is likely that implementations may include an array of hash algorithms, both weaker and stronger. The choice of hash algorithm for a given link will be based on the balance between strength of the hash and computation overhead. HMAC-MD5 is required as a baseline to be universally included in NodeOS implementations providing cryptographic authentication, with other proposals optional (see Section 6 on Conformance Requirements).

2. Data Structures

2.1. ANEP INTEGRITY Option Format

ANEP packets have options fields in the headers and a new INTEGRITY option is defined in this document.

The INTEGRITY option has the following format:



- *Key Identifier*: An unsigned 64-bit number that MUST be unique for a given sender. Locally unique Key Identifiers can be generated using some combination of the address (IP or MAC) of the sending interface and the key number. The combination of the Key Identifier and the sending system's network address uniquely identifies the security association (Section 2.2).

- *Sequence Number*: An unsigned 64-bit monotonically increasing, unique sequence number.

Sequence Number values may be any monotonically increasing sequence that provides the INTEGRITY option with a tag that is unique for the associated key's lifetime. Details on sequence number generation are presented in Section 3.

- *Keyed Message Digest*: The digest MUST be a multiple of 4 octets long. For HMAC-MD5, it will be 16 bytes long.

2.2. Security Association

The sending and receiving systems maintain a security association for each authentication key that they share. This security association includes the following parameters:

- Authentication algorithm and algorithm mode being used.
- Key used with the authentication algorithm.
- Lifetime of the key.
- Associated sending interface and other security association selection criteria [REQUIRED at Sending System].
- Source Address of the sending system [REQUIRED at Receiving System].
- Latest sending sequence number used with this key identifier [REQUIRED at Sending System].
- List of last N sequence numbers received with this key identifier [REQUIRED at Receiving System].

3. Generating Sequence Numbers

In this section we describe methods that could be chosen to generate the sequence numbers used in the INTEGRITY option of an ANEP packet. As previous stated, there are two important properties that MUST be satisfied by the generation procedure. The first property is that the sequence numbers are unique, or one-time, for the lifetime of the integrity key that is in current use. A receiver can use this property to unambiguously

distinguish between a new or a replayed message. The second property is that the sequence numbers are generated in monotonically increasing order, modulo 2^64. This is required to greatly reduce the amount of saved state, since a receiver only needs to save the value of the highest sequence number seen to avoid a replay attack. Since the starting sequence number might be arbitrarily large, the modulo operation is required to accommodate sequence number roll-over within some key's lifetime. This solution draws from TCP's approach [4].

The sequence number field is chosen to be a 64-bit unsigned quantity. This is large enough to avoid exhaustion over the key lifetime. For example, if a key lifetime was conservatively defined as one year, there would be enough sequence number values to send ANEP packets at an average rate of about 585 gigaPackets per second. A 32-bit sequence number would limit this average rate to about 136 packets per second.

The ability to generate unique monotonically increasing sequence numbers across a failure and restart implies some form of stable storage, either local to the device or remotely over the network. Three sequence number generation procedures are described below.

3.1. Simple Sequence Numbers

The most straightforward approach is to generate a unique sequence number using a message counter. Each time a message is transmitted for a given key, the sequence number counter is incremented. The current value of this counter is continually or periodically saved to stable storage. After a restart, the counter is recovered using this stable storage. If the counter was saved periodically to stable storage, the count should be recovered by increasing the saved value to be larger than any possible value of the counter at the time of the failure. This can be computed, knowing the interval at which the counter was saved to stable storage and incrementing the stored value by that amount.

3.2. Sequence Numbers Based on a Real Time Clock

Most devices will probably not have the capability to save sequence number counters to stable storage for each key. A more universal solution is to base sequence numbers on the stable storage of a real time clock. Many computing devices have a real time clock module that includes stable storage of the clock. These modules generally include some form of nonvolatile memory to retain clock information in the event of a power failure.

In this approach, we could use an NTP based timestamp value as the sequence number. The roll-over period of an NTP timestamp is about 136 years, much longer than any

reasonable lifetime of a key. In addition, the granularity of the NTP timestamp is fine enough to allow the generation of a message every 200 picoseconds for a given key. Many real time clock modules do not have the resolution of an NTP timestamp. In these cases, the least significant bits of the timestamp can be generated using a message counter, which is reset every clock tick. For example, when the real time clock provides a resolution of 1 second, the 32 least significant bits of the sequence number can be generated using a message counter. The remaining 32 bits are filled with the 32 least significant bits of the timestamp. Assuming that the recovery time after failure takes longer than one tick of the real time clock, the message counter for the low order bits can be safely reset to zero after a restart.

3.3. Sequence Numbers Based on a Network Recovered Clock

If the device does not contain any stable storage of sequence number counters or of a real time clock, it could recover the real time clock from the network using NTP. Once the clock has been recovered following a restart, the sequence number generation procedure would be identical to the procedure described above.

4. Message Processing

Implementations SHOULD allow specification of interfaces that are to be secured, for either sending messages, or receiving them, or both. The sender must ensure that all packets sent on secured sending interfaces include an INTEGRITY option, generated using the appropriate Key. Receivers verify whether ANEP packets, except of the type “Integrity Challenge” (Section 4.3), arriving on a secured receiving interface contain the INTEGRITY option. If the INTEGRITY option is absent, the receiver discards the packet.

Security associations are simplex - the keys that a sending system uses to sign its messages may be different from the keys that its receivers use to sign theirs. Hence, each association is associated with a unique sending system and (possibly) multiple receiving systems.

Each sender SHOULD have distinct security associations (and keys) per secured sending interface. While administrators may configure all the nodes on a subnet (or for that matter, in their network) using a single security association, implementations MUST assume that each sender may send using a distinct security association on each secured interface. At the sender, security association selection is based on the interface through which the message is sent. This selection MAY include additional criteria, such as the destination

address. Finally, all intended message recipients should participate in this security association. This is especially important for multicast messages.

Receivers select keys based on the Key Identifier and the sending system's network address. The Key Identifier is included in the INTEGRITY option. The sending system's address can be obtained by the use of the ANEP Source Identifier option, or if that's not present, from interface information that unambiguously identifies the sender (e.g. point to point interfaces). Since the Key Identifier is unique for a sender, this method uniquely identifies the key.

The integrity mechanism modifies the standard processing rules for ANEP packets, both when including the INTEGRITY option sent over a secured sending interface and when accepting a packet received on a secured receiving interface. These modifications are detailed below.

4.1. Message Generation

For an ANEP packet sent over a secured sending interface, the following steps must be applied:

(1) The INTEGRITY option is inserted in the appropriate place, and its location in the message is remembered for later use.

(2) The sending interface and other appropriate criteria (as mentioned above) are used to determine the Authentication Key and the hash algorithm to be used.

(3) The Source Identifier option is inserted in the appropriate place, if needed, and is set to the sender interface address.

(4) The sending sequence number MUST be updated to ensure a unique, monotonically increasing number. It is then placed in the Sequence Number field of the INTEGRITY option.

(5) The Keyed Message Digest field is set to zero.

(6) The Key Identifier is placed into the INTEGRITY option.

(7) An authenticating digest of the message is computed using the Authentication Key in conjunction with the keyed-hash algorithm. When the HMAC-MD5

algorithm is used, the hash calculation is described in [2]. The hash MUST include the entire message, the ANEP header, including all options, and any portion of the link layer headers which can be used by the receiving system to classify and demultiplex packets into NodeOS Channels.

(8) The digest is written into the Cryptographic Digest field of the INTEGRITY option.

4.2. Message Reception

When the message is received on a secured receiving interface, and is not of the type "Integrity Challenge", it is processed in the following manner:

(1) The Cryptographic Digest field of the INTEGRITY option is saved and the field is subsequently set to zero.

(2) The Key Identifier field and the sending system address are used to uniquely determine the Authentication Key and the hash algorithm to be used. Processing of this packet might be delayed when the Key Management System (Appendix 1) is queried for this information.

(3) A new keyed-digest is calculated using the indicated algorithm and the Authentication Key.

(4) If the calculated digest does not match the received digest, the message is discarded without further processing.

(5) If the message is of type "Integrity Response", verify that the CHALLENGE object identically matches the originated challenge. If it matches, save the sequence number in the INTEGRITY option as the largest sequence number received to date.

Otherwise, for all other ANEP packets, the sequence number is validated to prevent replay attacks, and messages with invalid sequence numbers are ignored by the receiver.

When a message is accepted, the sequence number of that message could update a stored value corresponding to the largest sequence number received to date. Each subsequent message must then have a larger (modulo 2^64)

sequence number to be accepted. This simple processing rule prevents message replay attacks, but it must be modified to tolerate limited out-of-order message delivery. For example, if several messages were sent in a burst by a node, they might get reordered and then the sequence numbers would not be received in an increasing order.

An implementation SHOULD allow administrative configuration that sets the receiver's tolerance to out-of-order message delivery. A simple approach would allow administrators to specify a message window corresponding to the worst case reordering behavior. For example, one might specify that packets reordered within a 32 message window would be accepted. If no reordering can occur, the window is set to one.

The receiver must store a list of all sequence numbers seen within the reordering window. A received sequence number is valid if (a) it is greater than the maximum sequence number received or (b) it is a past sequence number lying within the reordering window and not recorded in the list. Acceptance of a sequence number implies adding it to the list and removing a number from the lower end of the list. Messages received with sequence numbers lying below the lower end of the list or marked seen in the list are discarded.

When an "Integrity Challenge" message is received on a secured sending interface it is processed in the following manner:

(1) An "Integrity Response" message is formed using the Challenge object received in the challenge message.

(2) The message is sent back to the receiver, based on the source address of the challenge message, using the "Message Generation" steps outlined above. The selection of the Authentication Key and the hash algorithm to be used is determined by the key identifier supplied in the challenge message.

4.3. Integrity Handshake at Restart or Initialization of the Receiver

To obtain the starting sequence number for a live Authentication Key, the receiver MUST initiate an integrity handshake with the sender. This handshake consists of a receiver's Challenge and the sender's Response, and may be either initiated during restart or postponed until a message signed with that key arrives.

Once the receiver has decided to initiate an integrity handshake for a particular

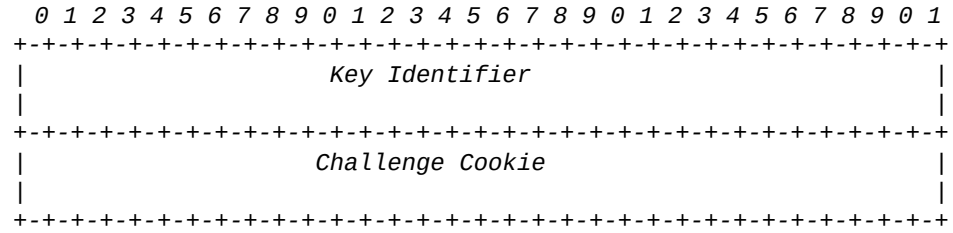
Authentication Key, it identifies the sender using the sending system's address configured in the corresponding security association. The receiver then sends an Integrity Challenge message to the sender. This message contains the Key Identifier to identify the sender's key and MUST have a unique challenge cookie that is based on a local secret to prevent guessing. see Section 2.5.3 of [4]). It is suggested that the cookie be an MD5 hash of a local secret and a timestamp to provide uniqueness (see Section 8).

[Author's note: The NodeOS group needs to identify ANEP type id(s) for use in NodeOS signaling activities. The handshake messages would then use these type id(s) and message structure. In the case of the Integrity Handshake, we need 2 distinct message types for the handshake.]

An Integrity Challenge message will carry a use a inter-NodeOS message with type number = ??. The message format is as follows:

```
<Integrity Challenge message> ::= <Common NodeOS Message Header>
                                  <CHALLENGE>
```

The CHALLENGE object has the following information:



The sender accepts the "Integrity Challenge" without doing an integrity check. It returns an "Integrity Response" message that contains the original CHALLENGE object. It also includes an INTEGRITY option, signed with the key specified by the Key Identifier included in the "Integrity Challenge".

THe Integrity Response message will carry a message type of ??. The message format is as follows:

```
<Integrity Response message> ::= <ANEP INTEGRITY option>
                                 <ANEP Source Identifier option>
                                 <Common NodeOS Message Header>
                                 <CHALLENGE>
```

The “Integrity Response” message is accepted by the receiver (challenger) only if the returned CHALLENGE object matches the one sent in the “Integrity Challenge” message. This prevents replay of old “Integrity Response” messages. If the match is successful, the receiver saves the Sequence Number from the INTEGRITY option as the latest sequence number received with the key identifier included in the CHALLENGE.

If a response is not received within a given period of time, the challenge is repeated. When the integrity handshake successfully completes, the receiver begins accepting normal ANEP messages from that sender and ignores any other “Integrity Response” messages.

5. Key Management

An integrated key management protocol was deliberately omitted from this specification. It would be desirable to use a key management protocol to distribute Authentication Keys among peering nodes. Such a protocol would provide scalability and significantly reduce the human administrative burden. When key management is deployed in the ABone, we expect that those protocols will be operational coupled to the mechanisms described in this document.

5.1. Key Management Procedures

Each key has a lifetime associated with it that is recorded in all systems (sender and receivers) configured with that key. The concept of a “key lifetime” merely requires that the earliest (KeyStartValid) and latest (KeyEndValid) times that the key is valid be programmable in a way the system understands. Certain key generation mechanisms, such as Kerberos or some public key schemes, may directly produce ephemeral keys. In this case, the lifetime of the key is implicitly defined as part of the key.

In general, no key is ever used outside its lifetime (but see Section 5.3). Possible mechanisms for managing key lifetime include the Network Time Protocol and hardware time-of-day clocks.

To maintain security, it is advisable to change the Authentication Key on a regular basis. It should be possible to switch the Authentication Key without loss of service and without requiring people to change all the keys at once. This requires an implementation to support the storage and use of more than one active Authentication Key at the same time. Hence both the sender and receivers might have multiple active keys for a given security association.

Since keys are shared between a sender and (possibly) multiple receivers, there is a region of uncertainty around the time of key switch-over during which some systems may still be using the old key and others might have switched to the new key. The size of this uncertainty region is related to clock synchrony of the systems. Administrators should configure the overlap between the expiration time of the old key (KeyEndValid) and the validity of the new key (KeyStartValid) to be at least twice the size of this uncertainty interval. This will allow the sender to make the key switch-over at the midpoint of this interval and be confident that all receivers are now accepting the new key. For the duration of the overlap in key lifetimes, a receiver must be prepared to authenticate messages using either key.

During a key switch-over, it will be necessary for each receiver to handshake with the sender using the new key. As stated before, a receiver has the choice of initiating a handshake during the switchover or postponing the handshake until the receipt of a message using that key.

5.2. Key Management Requirements

Requirements on an implementation are as follows:

- An implementation MUST support the storage and use of more than one key at the same time, for both sending and receiving systems.

- An implementation MUST associate a specific lifetime (i.e., KeyStartValid and KeyEndValid) with each key and the corresponding Key Identifier.

- An implementation MUST support manual key distribution (e.g., the privileged user manually typing in the key, key lifetime, and key identifier on the console). The lifetime may be infinite.

- If more than one algorithm is supported, then the implementation MUST require that the algorithm be specified for each key at the time the other key information is entered.

- Keys that are out of date MAY be automatically deleted by the implementation.

- Manual deletion of active keys MUST also be supported.

- Key storage SHOULD persist across a system restart, warm or cold, to ease operational usage.

5.3. Pathological Case

It is possible that the last key for a given security association has expired. When this happens, it is unacceptable to revert to an unauthenticated condition, and not advisable to disrupt current service. Therefore, the system should send a "last authentication key expiration" notification to the network manager and treat the key as having an infinite lifetime until the lifetime is extended, the key is deleted by network management, or a new key is configured.

6. Conformance Requirements

To conform to this specification, an implementation MUST support all of its aspects. The HMAC-MD5 authentication algorithm defined in [2] MUST be implemented by all conforming implementations. A conforming implementation MAY also support other authentication algorithms such as NIST's Secure Hash Algorithm (SHA). Manual key distribution as described above MUST be supported by all conforming implementations. All implementations MUST support the smooth key roll over described under "Key Management Procedures."

Implementations SHOULD support a standard key management protocol for secure distribution of Authentication Keys once such a key management protocol is deployed in the ABone.

7. References

[1] Braden, B., Lindell, B., Berson, S., "A Proposed ABone Network Security Architecture". Work in Progress. http://www.isi.edu/abone/

[2] Krawczyk, Bellare, and Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, March 1996.

[3] Postel, Jon, "Transmission Control Protocol", RFC 793, September 1981.

[4] Maughan, D., Schertler, M., Schneider, M., and J. Turner, "Internet Security Association and Key Management Protocol (ISAKMP)", RFC 2408, November 1998.

8. Security Considerations

The quality of the security provided by this mechanism depends on the strength of the implemented authentication algorithms, the strength of the key being used, and the correct implementation of the security mechanism in all communicating NodeOS implementations. This mechanism also depends on the Authentication Keys being kept confidential by all parties. If any of these assumptions are incorrect or procedures are insufficiently secure, then no real security will be provided to the users of this mechanism.

While the handshake “Integrity Response” message is integrity-checked, the handshake “Integrity Challenge” message is not. This was done intentionally to avoid the case when both peering nodes do not have a starting sequence number for each other’s key. Consequently, they will each keep sending handshake “Integrity Challenge” messages that will be dropped by the other end. Moreover, requiring only the response to be integrity-checked eliminates a dependency on an security association in the opposite direction.

This, however, lets an intruder generate fake handshaking challenges with a certain challenge cookie. It could then save the response and attempt to play it against a receiver that is in recovery. If it was lucky enough to have guessed the challenge cookie used by the receiver at recovery time it could use the saved response. This response would be accepted, since it is properly signed, and would have a smaller sequence number for the sender because it was an old message. This opens the receiver up to replays. Still, it seems very difficult to exploit. It requires not only guessing the challenge cookie (which is based on a locally known secret) in advance, but also being able to masquerade as the receiver to generate a handshake “Integrity Challenge” with the proper network address and not being caught.

9. Authors’ Addresses

Bob Lindell
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
Phone: (310) 822-1511
Email: lindell@ISI.EDU

10. Appendix 1: Key Management Interface

This appendix describes a generic interface to Key Management. This description is at an abstract level realizing that implementations may need to introduce small variations to the actual interface.

At the start of execution, a NodeOS would use this interface to obtain the current set of relevant keys for sending and receiving messages. During execution, a NodeOS can query for specific keys given a Key Identifier and Source Address, discover newly created keys, and be informed of those keys that have been deleted. The interface provides both a polling and asynchronous upcall style for wider applicability.

10.1. Data Structures

Information about keys is returned using the following KeyInfo data structure:

```
KeyInfo {
      Key Type (Send or Receive)
      KeyIdentifier
      Key
      Authentication Algorithm Type and Mode
      KeyStartValid
      KeyEndValid
      Status (Active or Deleted)
      Outgoing Interface (for Send only)
      Other Outgoing Security Association Selection Criteria
            (for Send only, optional)
      Sending System Address (for Receive Only)
}
```

10.2. Default Key Table

This function returns a list of KeyInfo data structures corresponding to all of the keys that are configured for sending and receiving ANEP messages and have an Active Status. This function is usually called at the start of execution but there is no limit on the number of times that it may be called.

```
KM_DefaultKeyTable() -> KeyInfoList
```

10.3. Querying for Unknown Receive Keys

When a message arrives with an unknown Key Identifier and Sending System Address pair, the NodeOS can use this function to query the Key Management System for the appropriate key. The status of the element returned, if any, must be Active.

```
KM_GetRecvKey( INTEGRITY Object, SrcAddress ) -> KeyInfo
```

10.4. Polling for Updates

This function returns a list of KeyInfo data structures corresponding to any incremental changes that have been made to the default key table or requested keys since the last call to either KM_KeyTablePoll, KM_DefaultKeyTable, or KM_GetRecvKey. The status of some elements in the returned list may be set to Deleted.

```
KM_KeyTablePoll() -> KeyInfoList
```

10.5. Asynchronous Upcall Interface

Rather than repeatedly calling the KM_KeyTablePoll(), an implementation may choose to use an asynchronous event model. This function registers interest to key changes for a given Key Identifier or for all keys if no Key Identifier is specified. The upcall function is called each time a change is made to a key.

```
KM_KeyUpdate ( Function [, KeyIdentifier ] )
```

where the upcall function is parameterized as follows:

```
Function ( KeyInfo )
```