Network Working Group
Request for Comments: 5590
Updates: 3411, 3412, 3414, 3417
Category: Standards Track

D. Harrington
Huawei Technologies (USA)
J. Schoenwaelder
Jacobs University Bremen

June 2009

Transport Subsystem for the Simple Network Management Protocol (SNMP)

Status of This Memo

Copyright Notice

Abstract

This document defines a Transport Subsystem, extending the Simple Network Management Protocol (SNMP) architecture defined in RFC 3411. This document defines a subsystem to contain Transport Models that is comparable to other subsystems in the RFC 3411 architecture. As work is being done to expand the transports to include secure transports, such as the Secure Shell (SSH) Protocol and Transport Layer Security

(TLS), using a subsystem will enable consistent design and modularity of such Transport Models. This document identifies and describes some key aspects that need to be considered for any Transport Model for SNMP.

Table of Contents

1. Introduction

This document defines a Transport Subsystem, extending the Simple Network Management Protocol (SNMP) architecture defined in [RFC3411]. This document identifies and describes some key aspects that need to be considered for any Transport Model for SNMP.

1.1. The Internet-Standard Management Framework

For a detailed overview of the documents that describe the current Internet-Standard Management Framework, please refer to Section 7 of RFC 3410 [RFC3410].

1.2. Conventions

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 RFC 2119 [RFC2119].

Lowercase versions of the keywords should be read as in normal English. They will usually, but not always, be used in a context that relates to compatibility with the RFC 3411 architecture or the subsystem defined here but that might have no impact on on-the-wire compatibility. These terms are used as guidance for designers of proposed IETF models to make the designs compatible with RFC 3411 subsystems and Abstract Service Interfaces (ASIs). Implementers are free to implement differently. Some usages of these lowercase terms are simply normal English usage.

For consistency with SNMP-related specifications, this document favors terminology as defined in STD 62, rather than favoring terminology that is consistent with non-SNMP specifications that use different variations of the same terminology. This is consistent with the IESG decision to not require the SNMPv3 terminology be modified to match the usage of other non-SNMP specifications when SNMPv3 was advanced to Full Standard.

This document discusses an extension to the modular RFC 3411 architecture; this is not a protocol document. An architectural "MUST" is a really sharp constraint; to allow for the evolution of technology and to not unnecessarily constrain future models, often a

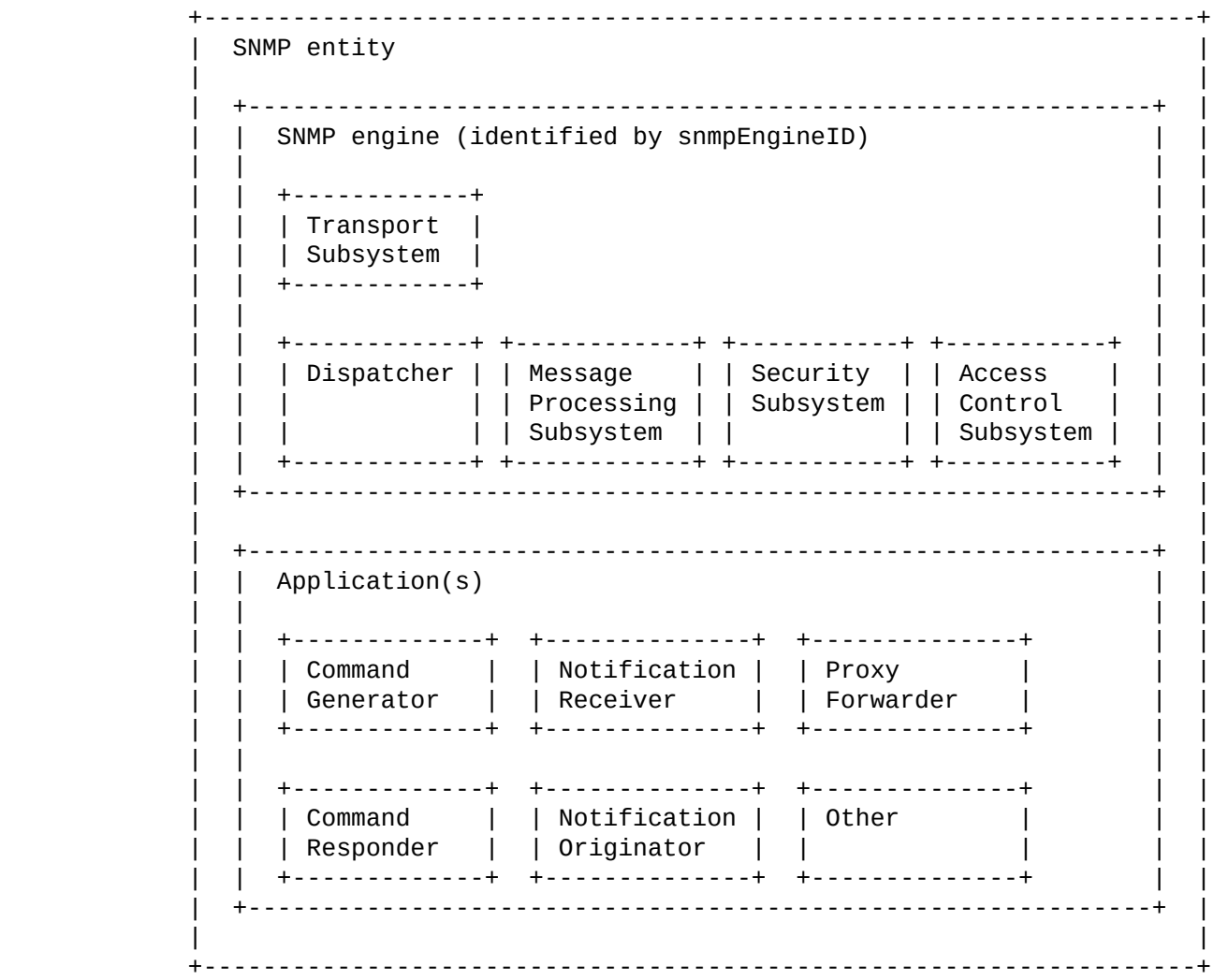
"SHOULD" or a "should" is more appropriate than a "MUST" in an architecture. Future models MAY express tighter requirements for their own model-specific processing.

1.3. Where This Extension Fits

It is expected that readers of this document will have read RFCs 3410 and 3411, and have a general understanding of the functionality defined in RFCs 3412-3418.

The "Transport Subsystem" is an additional component for the SNMP Engine depicted in RFC 3411, Section 3.1.

The following diagram depicts its place in the RFC 3411 architecture.



The transport mappings defined in RFC 3417 do not provide lower-layer security functionality, and thus do not provide transport-specific security parameters. This document updates RFC 3411 and RFC 3417 by defining an architectural extension and modifying the ASIs that transport mappings (hereafter called "Transport Models") can use to pass transport-specific security parameters to other subsystems, including transport-specific security parameters that are translated into the transport-independent securityName and securityLevel parameters.

The Transport Security Model [RFC5591] and the Secure Shell Transport Model [RFC5592] utilize the Transport Subsystem. The Transport Security Model is an alternative to the existing SNMPv1 Security Model [RFC3584], the SNMPv2c Security Model [RFC3584], and the User-based Security Model [RFC3414]. The Secure Shell Transport Model is an alternative to existing transport mappings as described in [RFC3417].

2. Motivation

Just as there are multiple ways to secure one's home or business, in a continuum of alternatives, there are multiple ways to secure a network management protocol. Let's consider three general approaches.

In the first approach, an individual could sit on his front porch waiting for intruders. In the second approach, he could hire an employee, schedule the employee, position the employee to guard what he wants protected, hire a second guard to cover if the first gets sick, and so on. In the third approach, he could hire a security company, tell them what he wants protected, and leave the details to them. Considerations of hiring and training employees, positioning and scheduling the guards, arranging for cover, etc., are the responsibility of the security company. The individual therefore achieves the desired security, with significantly less effort on his part except for identifying requirements and verifying the quality of service being provided.

The User-based Security Model (USM) as defined in [RFC3414] largely uses the first approach -- it provides its own security. It utilizes existing mechanisms (e.g., SHA), but provides all the coordination. USM provides for the authentication of a principal, message encryption, data integrity checking, timeliness checking, etc.

USM was designed to be independent of other existing security infrastructures. USM therefore uses a separate principal and key management infrastructure. Operators have reported that deploying another principal and key management infrastructure in order to use

SNMPv3 is a deterrent to deploying SNMPv3. It is possible to use external mechanisms to handle the distribution of keys for use by USM. The more important issue is that operators wanted to leverage existing user management infrastructures that were not specific to SNMP.

A USM-compliant architecture might combine the authentication mechanism with an external mechanism, such as RADIUS [RFC2865], to provide the authentication service. Similarly, it might be possible to utilize an external protocol to encrypt a message, to check timeliness, to check data integrity, etc. However, this corresponds to the second approach -- requiring the coordination of a number of differently subcontracted services. Building solid security between the various services is difficult, and there is a significant potential for gaps in security.

An alternative approach might be to utilize one or more lower-layer security mechanisms to provide the message-oriented security services required. These would include authentication of the sender, encryption, timeliness checking, and data integrity checking. This corresponds to the third approach described above. There are a number of IETF standards available or in development to address these problems through security layers at the transport layer or application layer, among them are TLS [RFC5246], Simple Authentication and Security Layer (SASL) [RFC4422], and SSH [RFC4251]

From an operational perspective, it is highly desirable to use security mechanisms that can unify the administrative security management for SNMPv3, command line interfaces (CLIs), and other management interfaces. The use of security services provided by lower layers is the approach commonly used for the CLI, and is also the approach being proposed for other network management protocols, such as syslog [RFC5424] and NETCONF [RFC4741].

This document defines a Transport Subsystem extension to the RFC 3411 architecture that is based on the third approach. This extension specifies how other lower-layer protocols with common security infrastructures can be used underneath the SNMP protocol and the desired goal of unified administrative security can be met.

This extension allows security to be provided by an external protocol connected to the SNMP engine through an SNMP Transport Model [RFC3417]. Such a Transport Model would then enable the use of existing security mechanisms, such as TLS [RFC5246] or SSH [RFC4251], within the RFC 3411 architecture.

There are a number of Internet security protocols and mechanisms that are in widespread use. Many of them try to provide a generic infrastructure to be used by many different application-layer protocols. The motivation behind the Transport Subsystem is to leverage these protocols where it seems useful.

There are a number of challenges to be addressed to map the security provided by a secure transport into the SNMP architecture so that SNMP continues to provide interoperability with existing implementations. These challenges are described in detail in this document. For some key issues, design choices are described that might be made to provide a workable solution that meets operational requirements and fits into the SNMP architecture defined in [RFC3411].

3. Requirements of a Transport Model

3.1. Message Security Requirements

Transport security protocols SHOULD provide protection against the following message-oriented threats:

1. modification of information

2. masquerade

3. message stream modification

4. disclosure

These threats are described in Section 1.4 of [RFC3411]. The security requirements outlined there do not require protection against denial of service or traffic analysis; however, transport security protocols should not make those threats significantly worse.

3.1.1. Security Protocol Requirements

There are a number of standard protocols that could be proposed as possible solutions within the Transport Subsystem. Some factors should be considered when selecting a protocol.

Using a protocol in a manner for which it was not designed has numerous problems. The advertised security characteristics of a protocol might depend on it being used as designed; when used in other ways, it might not deliver the expected security characteristics. It is recommended that any proposed model include a description of the applicability of the Transport Model.

A Transport Model SHOULD NOT require modifications to the underlying protocol. Modifying the protocol might change its security characteristics in ways that could impact other existing usages. If a change is necessary, the change SHOULD be an extension that has no impact on the existing usages. Any Transport Model specification should include a description of potential impact on other usages of the protocol.

Since multiple Transport Models can exist simultaneously within the Transport Subsystem, Transport Models MUST be able to coexist with each other.

3.2. SNMP Requirements

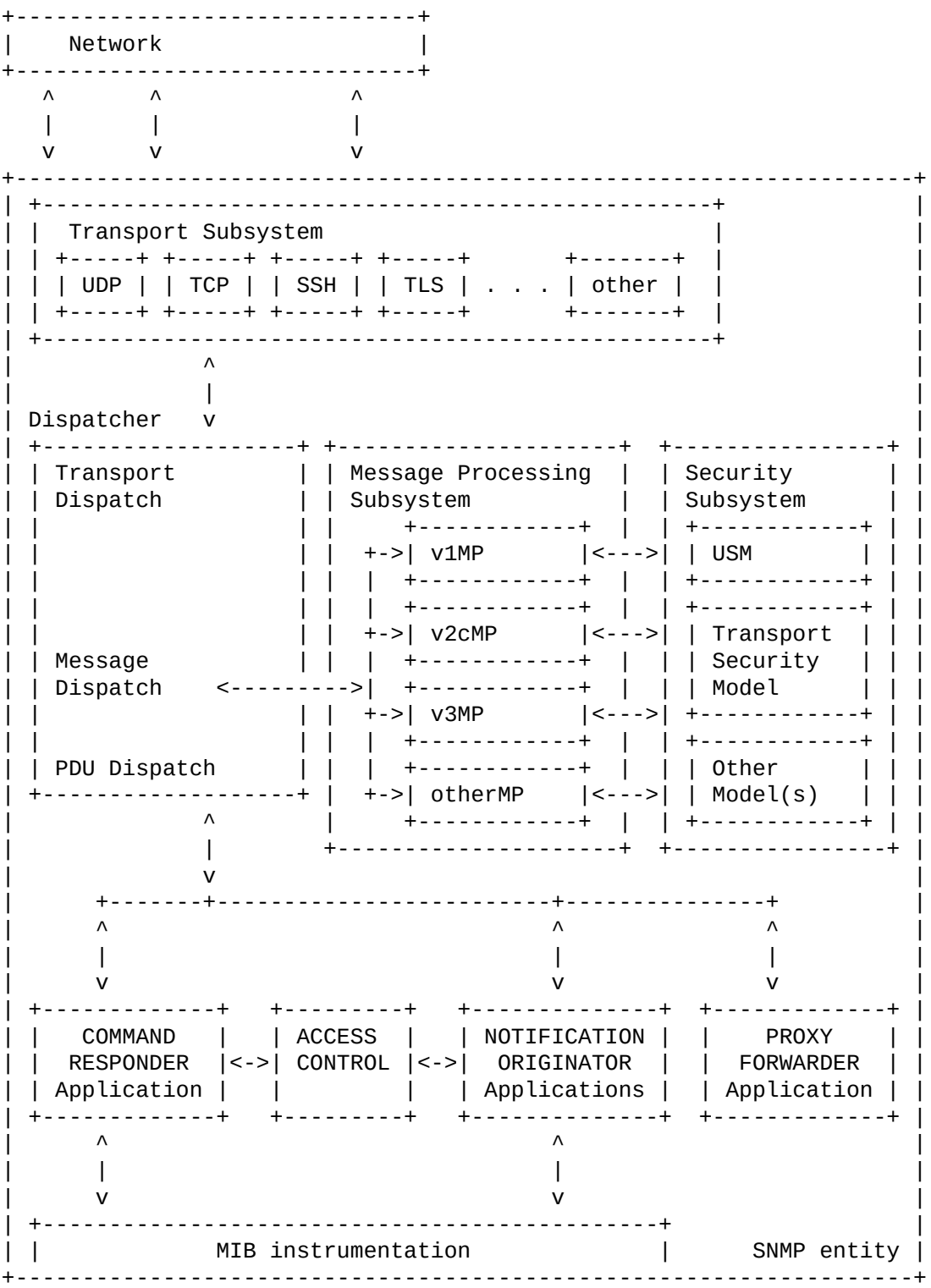
3.2.1. Architectural Modularity Requirements

SNMP version 3 (SNMPv3) is based on a modular architecture (defined in Section 3 of [RFC3411]) to allow the evolution of the SNMP protocol standards over time and to minimize the side effects between subsystems when changes are made.

The RFC 3411 architecture includes a Message Processing Subsystem for permitting different message versions to be handled by a single engine, a Security Subsystem for enabling different methods of providing security services, Applications to support different types of Application processors, and an Access Control Subsystem for allowing multiple approaches to access control. The RFC 3411 architecture does not include a subsystem for Transport Models, despite the fact there are multiple transport mappings already defined for SNMP [RFC3417]. This document describes a Transport Subsystem that is compatible with the RFC 3411 architecture. As work is being done to use secure transports such as SSH and TLS, using a subsystem will enable consistent design and modularity of such Transport Models.

The design of this Transport Subsystem accepts the goals of the RFC 3411 architecture that are defined in Section 1.5 of [RFC3411]. This Transport Subsystem uses a modular design that permits Transport Models (which might or might not be security-aware) to be "plugged into" the RFC 3411 architecture. Such Transport Models would be independent of other modular SNMP components as much as possible. This design also permits Transport Models to be advanced through the standards process independently of other Transport Models.

The following diagram depicts the SNMPv3 architecture, including the new Transport Subsystem defined in this document and a new Transport Security Model defined in [RFC5591].


Network
Transport Subsystem
UDP
TCP
SSH
TLS
. . .
other
Dispatcher
Transport Dispatch
Message Dispatch
PDU Dispatch
Message Processing Subsystem
v1MP
v2cMP
v3MP
otherMP
Security Subsystem
USM
Transport Security Model
Other Model(s)
COMMAND RESPONDER Application
ACCESS CONTROL
NOTIFICATION ORIGINATOR Applications
PROXY FORWARDER Application
MIB instrumentation
SNMP entity

3.2.1.1. Changes to the RFC 3411 Architecture

The RFC 3411 architecture and the Security Subsystem assume that a Security Model is called by a Message Processing Model and will perform multiple security functions within the Security Subsystem. A Transport Model that supports a secure transport protocol might perform similar security functions within the Transport Subsystem, including the translation of transport-security parameters to/from Security-Model-independent parameters.

To accommodate this, an implementation-specific cache of transport-specific information will be described (not shown), and the data flows on this path will be extended to pass Security-Model-independent values. This document amends some of the ASIs defined in RFC 3411; these changes are covered in Section 6 of this document.

New Security Models might be defined that understand how to work with these modified ASIs and the transport-information cache. One such Security Model, the Transport Security Model, is defined in [RFC5591].

3.2.1.2. Changes to RFC 3411 Processing

The introduction of secure transports affects the responsibilities and order of processing within the RFC 3411 architecture. While the steps are the same, they might occur in a different order, and might be done by different subsystems. With the existing RFC 3411 architecture, security processing starts when the Message Processing Model decodes portions of the encoded message to extract parameters that identify which Security Model MUST handle the security-related tasks.

A secure transport performs those security functions on the message, before the message is decoded. Some of these functions might then be repeated by the selected Security Model.

3.2.1.3. Passing Information between SNMP Engines

A secure Transport Model will establish an authenticated and possibly encrypted tunnel between the Transport Models of two SNMP engines. After a transport-layer tunnel is established, then SNMP messages can be sent through the tunnel from one SNMP engine to the other. While the Community Security Models [RFC3584] and the User-based Security Model establish a security association for each SNMP message, newer Transport Models MAY support sending multiple SNMP messages through the same tunnel to amortize the costs of establishing a security association.

3.2.2. Access Control Requirements

RFC 3411 made some design decisions related to the support of an Access Control Subsystem. These include establishing and passing in a model-independent manner the securityModel, securityName, and securityLevel parameters, and separating message authentication from data-access authorization.

3.2.2.1. securityName and securityLevel Mapping

SNMP data-access controls are expected to work on the basis of who can perform what operations on which subsets of data, and based on the security services that will be provided to secure the data in transit. The securityModel and securityLevel parameters establish the protections for transit -- whether authentication and privacy services will be or have been applied to the message. The securityName is a model-independent identifier of the security "principal".

A Security Model plays a role in security that goes beyond protecting the message -- it provides a mapping between the Security-Model-specific principal for an incoming message to a Security-Model independent securityName that can be used for subsequent processing, such as for access control. The securityName is mapped from a mechanism-specific identity, and this mapping must be done for incoming messages by the Security Model before it passes securityName to the Message Processing Model via the processIncoming ASI.

A Security Model is also responsible to specify, via the securityLevel parameter, whether incoming messages have been authenticated and encrypted, and to ensure that outgoing messages are authenticated and encrypted based on the value of securityLevel.

A Transport Model MAY provide suggested values for securityName and securityLevel. A Security Model might have multiple sources for determining the principal and desired security services, and a particular Security Model might or might not utilize the values proposed by a Transport Model when deciding the value of securityName and securityLevel.

Documents defining a new transport domain MUST define a prefix that MAY be prepended to all securityNames passed by the Security Model. The prefix MUST include one to four US-ASCII alpha-numeric characters, not including a ":" (US-ASCII 0x3a) character. If a prefix is used, a securityName is constructed by concatenating the prefix and a ":" (US-ASCII 0x3a) character, followed by a non-empty identity in an snmpAdminString-compatible format. The prefix can be used by SNMP Applications to distinguish "alice" authenticated by SSH

from "alice" authenticated by TLS. Transport domains and their corresponding prefixes are coordinated via the IANA registry "SNMP Transport Domains".

3.2.3. Security Parameter Passing Requirements

A Message Processing Model might unpack SNMP-specific security parameters from an incoming message before calling a specific Security Model to handle the security-related processing of the message. When using a secure Transport Model, some security parameters might be extracted from the transport layer by the Transport Model before the message is passed to the Message Processing Subsystem.

This document describes a cache mechanism (see Section 5) into which the Transport Model puts information about the transport and security parameters applied to a transport connection or an incoming message; a Security Model might extract that information from the cache. A tmStateReference is passed as an extra parameter in the ASIs between the Transport Subsystem and the Message Processing and Security Subsystems in order to identify the relevant cache. This approach of passing a model-independent reference is consistent with the securityStateReference cache already being passed around in the RFC 3411 ASIs.

3.2.4. Separation of Authentication and Authorization

The RFC 3411 architecture defines a separation of authentication and the authorization to access and/or modify MIB data. A set of model-independent parameters (securityModel, securityName, and securityLevel) are passed between the Security Subsystem, the Applications, and the Access Control Subsystem.

This separation was a deliberate decision of the SNMPv3 WG, in order to allow support for authentication protocols that do not provide data-access authorization capabilities, and in order to support data-access authorization schemes, such as the View-based access Control Model (VACM), that do not perform their own authentication.

A Message Processing Model determines which Security Model is used, either based on the message version (e.g., SNMPv1 and SNMPv2c) or possibly by a value specified in the message (e.g., msgSecurityModel field in SNMPv3).

The Security Model makes the decision which securityName and securityLevel values are passed as model-independent parameters to an Application, which then passes them via the isAccessAllowed ASI to the Access Control Subsystem.

An Access Control Model performs the mapping from the model-independent security parameters to a policy within the Access Control Model that is Access-Control-Model-dependent.

A Transport Model does not know which Security Model will be used for an incoming message, and so cannot know how the securityName and securityLevel parameters will be determined. It can propose an authenticated identity (via the tmSecurityName field), but there is no guarantee that this value will be used by the Security Model. For example, non-transport-aware Security Models will typically determine the securityName (and securityLevel) based on the contents of the SNMP message itself. Such Security Models will simply not know that the tmStateReference cache exists.

Further, even if the Transport Model can influence the choice of securityName, it cannot directly determine the authorization allowed to this identity. If two different Transport Models each authenticate a transport principal that are then both mapped to the same securityName, then these two identities will typically be afforded exactly the same authorization by the Access Control Model.

The only way for the Access Control Model to differentiate between identities based on the underlying Transport Model would be for such transport-authenticated identities to be mapped to distinct securityNames. How and if this is done is Security-Model-dependent.

3.3. Session Requirements

Some secure transports have a notion of sessions, while other secure transports provide channels or other session-like mechanisms. Throughout this document, the term "session" is used in a broad sense to cover transport sessions, transport channels, and other transport-layer, session-like mechanisms. Transport-layer sessions that can secure multiple SNMP messages within the lifetime of the session are considered desirable because the cost of authentication can be amortized over potentially many transactions. How a transport session is actually established, opened, closed, or maintained is specific to a particular Transport Model.

To reduce redundancy, this document describes aspects that are expected to be common to all Transport Model sessions.

3.3.1. No SNMP Sessions

The architecture defined in [RFC3411] and the Transport Subsystem defined in this document do not support SNMP sessions or include a session selector in the Abstract Service Interfaces.

The Transport Subsystem might support transport sessions. However, the Transport Subsystem does not have access to the pduType (i.e., the SNMP operation type), and so cannot select a given transport session for particular types of traffic.

Certain parameters of the Abstract Service Interfaces might be used to guide the selection of an appropriate transport session to use for a given request by an Application.

The transportDomain and transportAddress identify the transport connection to a remote network node. Elements of the transport address (such as the port number) might be used by an Application to send a particular PDU type to a particular transport address. For example, the SNMP-TARGET-MIB and SNMP-NOTIFICATION-MIB [RFC3413] are used to configure notification originators with the destination port to which SNMPv2-Trap PDUs or Inform PDUs are to be sent, but the Transport Subsystem never looks inside the PDU.

The securityName identifies which security principal to communicate with at that address (e.g., different Network Management System (NMS) applications), and the securityLevel might permit selection of different sets of security properties for different purposes (e.g., encrypted SET vs. non-encrypted GET operations).

However, because the handling of transport sessions is specific to each Transport Model, some Transport Models MAY restrict selecting a particular transport session. A user application might use a unique combination of transportDomain, transportAddress, securityModel, securityName, and securityLevel to try to force the selection of a given transport session. This usage is NOT RECOMMENDED because it is not guaranteed to be interoperable across implementations and across models.

Implementations SHOULD be able to maintain some reasonable number of concurrent transport sessions, and MAY provide non-standard internal mechanisms to select transport sessions.

3.3.2. Session Establishment Requirements

SNMP Applications provide the transportDomain, transportAddress, securityName, and securityLevel to be used to create a new session.

If the Transport Model cannot provide at least the requested level of security, the Transport Model should discard the message and should notify the Dispatcher that establishing a session and sending the message failed. Similarly, if the session cannot be established, then the message should be discarded and the Dispatcher notified.

Transport session establishment might require provisioning authentication credentials at an engine, either statically or dynamically. How this is done is dependent on the Transport Model and the implementation.

3.3.3. Session Maintenance Requirements

A Transport Model can tear down sessions as needed. It might be necessary for some implementations to tear down sessions as the result of resource constraints, for example.

The decision to tear down a session is implementation-dependent. How an implementation determines that an operation has completed is implementation-dependent. While it is possible to tear down each transport session after processing for each message has completed, this is not recommended for performance reasons.

The elements of procedure describe when cached information can be discarded, and the timing of cache cleanup might have security implications, but cache memory management is an implementation issue.

If a Transport Model defines MIB module objects to maintain session state information, then the Transport Model MUST define what happens to the objects when a related session is torn down, since this will impact the interoperability of the MIB module.

3.3.4. Message Security versus Session Security

A Transport Model session is associated with state information that is maintained for its lifetime. This state information allows for the application of various security services to multiple messages. Cryptographic keys associated with the transport session SHOULD be used to provide authentication, integrity checking, and encryption services, as needed, for data that is communicated during the session. The cryptographic protocols used to establish keys for a Transport Model session SHOULD ensure that fresh new session keys are generated for each session. This would ensure that a cross-session replay attack would be unsuccessful; that is, an attacker could not take a message observed on one session and successfully replay it on another session.

A good security protocol would also protect against replay attacks within a session; that is, an attacker could not take a message observed on a session and successfully replay it later in the same session. One approach would be to use sequence information within the protocol, allowing the participants to detect if messages were replayed or reordered within a session.

If a secure transport session is closed between the time a request message is received and the corresponding response message is sent, then the response message SHOULD be discarded, even if a new session has been established. The SNMPv3 WG decided that this should be a "SHOULD" architecturally, and it is a Security-Model-specific decision whether to REQUIRE this. The architecture does not mandate this requirement in order to allow for future Security Models where this might make sense; however, not requiring this could lead to added complexity and security vulnerabilities, so most Security Models SHOULD require this.

SNMPv3 was designed to support multiple levels of security, selectable on a per-message basis by an SNMP Application, because, for example, there is not much value in using encryption for a command generator to poll for potentially non-sensitive performance data on thousands of interfaces every ten minutes; such encryption might add significant overhead to processing of the messages.

Some Transport Models might support only specific authentication and encryption services, such as requiring all messages to be carried using both authentication and encryption, regardless of the security level requested by an SNMP Application. A Transport Model MAY upgrade the security level requested by a transport-aware Security Model, i.e., noAuthNoPriv and authNoPriv might be sent over an authenticated and encrypted session. A Transport Model MUST NOT downgrade the security level requested by a transport-aware Security Model, and SHOULD discard any message where this would occur. This is a SHOULD rather than a MUST only to permit the potential development of models that can perform error-handling in a manner that is less severe than discarding the message. However, any model that does not discard the message in this circumstance should have a clear justification for why not discarding will not create a security vulnerability.

4. Scenario Diagrams and the Transport Subsystem

Sections 4.6.1 and 4.6.2 of RFC 3411 provide scenario diagrams to illustrate how an outgoing message is created and how an incoming message is processed. RFC 3411 does not define ASIs for the "Send SNMP Request Message to Network", "Receive SNMP Response Message from Network", "Receive SNMP Message from Network" and "Send SNMP message to Network" arrows in these diagrams.

This document defines two ASIs corresponding to these arrows: a sendMessage ASI to send SNMP messages to the network and a receiveMessage ASI to receive SNMP messages from the network. These ASIs are used for all SNMP messages, regardless of pduType.

5. Cached Information and References

When performing SNMP processing, there are two levels of state information that might need to be retained: the immediate state linking a request-response pair and a potentially longer-term state relating to transport and security.

The RFC 3411 architecture uses caches to maintain the short-term message state, and uses references in the ASIs to pass this information between subsystems.

This document defines the requirements for a cache to handle additional short-term message state and longer-term transport state information, using a tmStateReference parameter to pass this information between subsystems.

To simplify the elements of procedure, the release of state information is not always explicitly specified. As a general rule, if state information is available when a message being processed gets discarded, the state related to that message should also be discarded. If state information is available when a relationship between engines is severed, such as the closing of a transport session, the state information for that relationship should also be discarded.

Since the contents of a cache are meaningful only within an implementation, and not on-the-wire, the format of the cache is implementation-specific.

5.1. securityStateReference

The securityStateReference parameter is defined in RFC 3411. Its primary purpose is to provide a mapping between a request and the corresponding response. This cache is not accessible to Transport Models, and an entry is typically only retained for the lifetime of a request-response pair of messages.

5.2. tmStateReference

For each transport session, information about the transport security is stored in a tmState cache or datastore that is referenced by a tmStateReference. The tmStateReference parameter is used to pass model-specific and mechanism-specific parameters between the Transport Subsystem and transport-aware Security Models.

In general, when necessary, the tmState is populated by the Security Model for outgoing messages and by the Transport Model for incoming messages. However, in both cases, the model populating the tmState

might have incomplete information, and the missing information might be populated by the other model when the information becomes available.

The tmState might contain both long-term and short-term information. The session information typically remains valid for the duration of the transport session, might be used for several messages, and might be stored in a local configuration datastore. Some information has a shorter lifespan, such as tmSameSecurity and tmRequestedSecurityLevel, which are associated with a specific message.

Since this cache is only used within an implementation, and not on-the-wire, the precise contents and format of the cache are implementation-dependent. For architectural modularity between Transport Models and transport-aware Security Models, a fully-defined tmState MUST conceptually include at least the following fields:

- tmTransportDomain
- tmTransportAddress
- tmSecurityName
- tmRequestedSecurityLevel
- tmTransportSecurityLevel
- tmSameSecurity
- tmSessionID

The details of these fields are described in the following subsections.

5.2.1. Transport Information

Information about the source of an incoming SNMP message is passed up from the Transport Subsystem as far as the Message Processing Subsystem. However, these parameters are not included in the processIncomingMsg ASI defined in RFC 3411; hence, this information is not directly available to the Security Model.

A transport-aware Security Model might wish to take account of the transport protocol and originating address when authenticating the request and setting up the authorization parameters. It is therefore

necessary for the Transport Model to include this information in the tmStateReference cache so that it is accessible to the Security Model.

- tmTransportDomain: the transport protocol (and hence the Transport Model) used to receive the incoming message.

- tmTransportAddress: the source of the incoming message.

The ASIs used for processing an outgoing message all include explicit transportDomain and transportAddress parameters. The values within the securityStateReference cache might override these parameters for outgoing messages.

5.2.2. securityName

There are actually three distinct "identities" that can be identified during the processing of an SNMP request over a secure transport:

- transport principal: the transport-authenticated identity on whose behalf the secure transport connection was (or should be) established. This value is transport-, mechanism-, and implementation-specific, and is only used within a given Transport Model.

- tmSecurityName: a human-readable name (in snmpAdminString format) representing this transport identity. This value is transport- and implementation-specific, and is only used (directly) by the Transport and Security Models.

- securityName: a human-readable name (in snmpAdminString format) representing the SNMP principal in a model-independent manner. This value is used directly by SNMP Applications, the Access Control Subsystem, the Message Processing Subsystem, and the Security Subsystem.

The transport principal might or might not be the same as the tmSecurityName. Similarly, the tmSecurityName might or might not be the same as the securityName as seen by the Application and Access Control Subsystems. In particular, a non-transport-aware Security Model will ignore tmSecurityName completely when determining the SNMP securityName.

However, it is important that the mapping between the transport principal and the SNMP securityName (for transport-aware Security Models) is consistent and predictable in order to allow configuration of suitable access control and the establishment of transport connections.

5.2.3. securityLevel

There are two distinct issues relating to security level as applied to secure transports. For clarity, these are handled by separate fields in the tmStateReference cache:

- tmTransportSecurityLevel: an indication from the Transport Model of the level of security offered by this session. The Security Model can use this to ensure that incoming messages were suitably protected before acting on them.

- tmRequestedSecurityLevel: an indication from the Security Model of the level of security required to be provided by the transport protocol. The Transport Model can use this to ensure that outgoing messages will not be sent over an insufficiently secure session.

5.2.4. Session Information

For security reasons, if a secure transport session is closed between the time a request message is received and the corresponding response message is sent, then the response message SHOULD be discarded, even if a new session has been established. The SNMPv3 WG decided that this should be a "SHOULD" architecturally, and it is a Security-Model-specific decision whether to REQUIRE this.

- tmSameSecurity: this flag is used by a transport-aware Security Model to indicate whether the Transport Model MUST enforce this restriction.

- tmSessionID: in order to verify whether the session has changed, the Transport Model must be able to compare the session used to receive the original request with the one to be used to send the response. This typically needs some form of session identifier. This value is only ever used by the Transport Model, so the format and interpretation of this field are model-specific and implementation-dependent.

When processing an outgoing message, if tmSameSecurity is true, then the tmSessionID MUST match the current transport session; otherwise, the message MUST be discarded and the Dispatcher notified that sending the message failed.

6. Abstract Service Interfaces

Abstract service interfaces have been defined by RFC 3411 to describe the conceptual data flows between the various subsystems within an SNMP entity and to help keep the subsystems independent of each other except for the common parameters.

This document introduces a couple of new ASIs to define the interface between the Transport and Dispatcher Subsystems; it also extends some of the ASIs defined in RFC 3411 to include transport-related information.

This document follows the example of RFC 3411 regarding the release of state information and regarding error indications.

1) The release of state information is not always explicitly specified in a Transport Model. As a general rule, if state information is available when a message gets discarded, the message-state information should also be released, and if state information is available when a session is closed, the session-state information should also be released. Keeping sensitive security information longer than necessary might introduce potential vulnerabilities to an implementation.

2)An error indication in statusInformation will typically include the Object Identifier (OID) and value for an incremented error counter. This might be accompanied by values for contextEngineID and contextName for this counter, a value for securityLevel, and the appropriate state reference if the information is available at the point where the error is detected.

6.1. sendMessage ASI

The sendMessage ASI is used to pass a message from the Dispatcher to the appropriate Transport Model for sending. The sendMessageASI defined in this document replaces the text "Send SNMP Request Message to Network" that appears in the diagram in Section 4.6.1 of RFC 3411 and the text "Send SNMP Message to Network" that appears in Section 4.6.2 of RFC 3411.

If present and valid, the tmStateReference refers to a cache containing Transport-Model-specific parameters for the transport and transport security. How a tmStateReference is determined to be present and valid is implementation-dependent. How the information in the cache is used is Transport-Model-dependent and implementation-dependent.

This might sound underspecified, but a Transport Model might be something like SNMP over UDP over IPv6, where no security is provided, so it might have no mechanisms for utilizing a tmStateReference cache.

```
statusInformation =
sendMessage(
IN   destTransportDomain           -- transport domain to be used
IN   destTransportAddress          -- transport address to be used
IN   outgoingMessage               -- the message to send
IN   outgoingMessageLength         -- its length
IN   tmStateReference              -- reference to transport state
 )
```

6.2. Changes to RFC 3411 Outgoing ASIs

Additional parameters have been added to the ASIs defined in RFC 3411 that are concerned with communication between the Dispatcher and Message Processing Subsystems, and between the Message Processing and Security Subsystems.

6.2.1. Message Processing Subsystem Primitives

A tmStateReference parameter has been added as an OUT parameter to the prepareOutgoingMessage and prepareResponseMessage ASIs. This is passed from the Message Processing Subsystem to the Dispatcher, and from there to the Transport Subsystem.

How or if the Message Processing Subsystem modifies or utilizes the contents of the cache is Message-Processing-Model specific.

```
statusInformation =          -- success or errorIndication
prepareOutgoingMessage(
IN  transportDomain          -- transport domain to be used
IN  transportAddress         -- transport address to be used
IN  messageProcessingModel   -- typically, SNMP version
IN  securityModel            -- Security Model to use
IN  securityName             -- on behalf of this principal
IN  securityLevel            -- Level of Security requested
IN  contextEngineID          -- data from/at this entity
IN  contextName              -- data from/in this context
IN  pduVersion               -- the version of the PDU
IN  PDU                      -- SNMP Protocol Data Unit
IN  expectResponse           -- TRUE or FALSE
IN  sendPduHandle            -- the handle for matching
                                incoming responses
```

```
   OUT  destTransportDomain     -- destination transport domain
   OUT  destTransportAddress    -- destination transport address
   OUT  outgoingMessage         -- the message to send
   OUT  outgoingMessageLength   -- its length
   OUT  tmStateReference        -- (NEW) reference to transport state
              )

   statusInformation =          -- success or errorIndication
   prepareResponseMessage(
   IN  messageProcessingModel   -- typically, SNMP version
   IN  securityModel            -- Security Model to use
   IN  securityName             -- on behalf of this principal
   IN  securityLevel            -- Level of Security requested
   IN  contextEngineID          -- data from/at this entity
   IN  contextName              -- data from/in this context
   IN  pduVersion               -- the version of the PDU
   IN  PDU                      -- SNMP Protocol Data Unit
   IN  maxSizeResponseScopedPDU -- maximum size able to accept
   IN  stateReference           -- reference to state information
                                -- as presented with the request
   IN  statusInformation        -- success or errorIndication
                                -- error counter OID/value if error
   OUT destTransportDomain      -- destination transport domain
   OUT destTransportAddress     -- destination transport address
   OUT outgoingMessage          -- the message to send
   OUT outgoingMessageLength    -- its length
   OUT tmStateReference         -- (NEW) reference to transport state
              )
```

6.2.2. Security Subsystem Primitives

transportDomain and transportAddress parameters have been added as IN parameters to the generateRequestMsg and generateResponseMsg ASIs, and a tmStateReference parameter has been added as an OUT parameter. The transportDomain and transportAddress parameters will have been passed into the Message Processing Subsystem from the Dispatcher and are passed on to the Security Subsystem. The tmStateReference parameter will be passed from the Security Subsystem back to the Message Processing Subsystem, and on to the Dispatcher and Transport Subsystems.

If a cache exists for a session identifiable from the tmTransportDomain, tmTransportAddress, tmSecurityName, and requested securityLevel, then a transport-aware Security Model might create a tmStateReference parameter to this cache and pass that as an OUT parameter.

```
statusInformation =
generateRequestMsg(
  IN   transportDomain         -- (NEW) destination transport domain
  IN   transportAddress        -- (NEW) destination transport address
  IN   messageProcessingModel  -- typically, SNMP version
  IN   globalData              -- message header, admin data
  IN   maxMessageSize          -- of the sending SNMP entity
  IN   securityModel           -- for the outgoing message
  IN   securityEngineID        -- authoritative SNMP entity
  IN   securityName            -- on behalf of this principal
  IN   securityLevel           -- Level of Security requested
  IN   scopedPDU               -- message (plaintext) payload
  OUT  securityParameters      -- filled in by Security Module
  OUT  wholeMsg                -- complete generated message
  OUT  wholeMsgLength          -- length of generated message
  OUT  tmStateReference        -- (NEW) reference to transport state
       )

statusInformation =
generateResponseMsg(
  IN   transportDomain         -- (NEW) destination transport domain
  IN   transportAddress        -- (NEW) destination transport address
  IN   messageProcessingModel -- Message Processing Model
  IN   globalData             -- msgGlobalData
  IN   maxMessageSize         -- from msgMaxSize
  IN   securityModel          -- as determined by MPM
  IN   securityEngineID       -- the value of snmpEngineID
  IN   securityName           -- on behalf of this principal
  IN   securityLevel          -- for the outgoing message
  IN   scopedPDU              -- as provided by MPM
  IN   securityStateReference -- as provided by MPM
  OUT  securityParameters     -- filled in by Security Module
  OUT  wholeMsg               -- complete generated message
  OUT  wholeMsgLength         -- length of generated message
  OUT  tmStateReference       -- (NEW) reference to transport state
       )
```

6.3. The receiveMessage ASI

The receiveMessage ASI is used to pass a message from the Transport Subsystem to the Dispatcher. The receiveMessage ASI replaces the text "Receive SNMP Response Message from Network" that appears in the diagram in Section 4.6.1 of RFC 3411 and the text "Receive SNMP Message from Network" from Section 4.6.2 of RFC3411.

When a message is received on a given transport session, if a cache does not already exist for that session, the Transport Model might create one, referenced by tmStateReference. The contents of this

cache are discussed in Section 5. How this information is determined is implementation- and Transport-Model-specific.

"Might create one" might sound underspecified, but a Transport Model might be something like SNMP over UDP over IPv6, where transport security is not provided, so it might not create a cache.

The Transport Model does not know the securityModel for an incoming message; this will be determined by the Message Processing Model in a Message-Processing-Model-dependent manner.

```
statusInformation =
receiveMessage(
IN   transportDomain              -- origin transport domain
IN   transportAddress             -- origin transport address
IN   incomingMessage              -- the message received
IN   incomingMessageLength        -- its length
IN   tmStateReference             -- reference to transport state
 )
```

6.4. Changes to RFC 3411 Incoming ASIs

The tmStateReference parameter has also been added to some of the incoming ASIs defined in RFC 3411. How or if a Message Processing Model or Security Model uses tmStateReference is message-processing- and Security-Model-specific.

This might sound underspecified, but a Message Processing Model might have access to all the information from the cache and from the message. The Message Processing Model might determine that the USM Security Model is specified in an SNMPv3 message header; the USM Security Model has no need of values in the tmStateReference cache to authenticate and secure the SNMP message, but an Application might have specified to use a secure transport such as that provided by the SSH Transport Model to send the message to its destination.

6.4.1. Message Processing Subsystem Primitive

The tmStateReference parameter of prepareDataElements is passed from the Dispatcher to the Message Processing Subsystem. How or if the Message Processing Subsystem modifies or utilizes the contents of the cache is Message-Processing-Model-specific.

```
result =                      -- SUCCESS or errorIndication
prepareDataElements(
IN   transportDomain          -- origin transport domain
IN   transportAddress         -- origin transport address
IN   wholeMsg                 -- as received from the network
```

```
   IN   wholeMsgLength             -- as received from the network
   IN   tmStateReference           -- (NEW) from the Transport Model
   OUT  messageProcessingModel     -- typically, SNMP version
   OUT  securityModel              -- Security Model to use
   OUT  securityName               -- on behalf of this principal
   OUT  securityLevel              -- Level of Security requested
   OUT  contextEngineID            -- data from/at this entity
   OUT  contextName                -- data from/in this context
   OUT  pduVersion                 -- the version of the PDU
   OUT  PDU                        -- SNMP Protocol Data Unit
   OUT  pduType                    -- SNMP PDU type
   OUT  sendPduHandle              -- handle for matched request
   OUT  maxSizeResponseScopedPDU   -- maximum size sender can accept
   OUT  statusInformation          -- success or errorIndication
                                   -- error counter OID/value if error
   OUT  stateReference             -- reference to state information
                                   -- to be used for possible Response
   )
```

6.4.2. Security Subsystem Primitive

The processIncomingMessage ASI passes tmStateReference from the Message Processing Subsystem to the Security Subsystem.

If tmStateReference is present and valid, an appropriate Security Model might utilize the information in the cache. How or if the Security Subsystem utilizes the information in the cache is Security-Model-specific.

```
   statusInformation =  -- errorIndication or success
                            -- error counter OID/value if error
   processIncomingMsg(
   IN   messageProcessingModel     -- typically, SNMP version
   IN   maxMessageSize             -- of the sending SNMP entity
   IN   securityParameters         -- for the received message
   IN   securityModel              -- for the received message
   IN   securityLevel              -- Level of Security
   IN   wholeMsg                   -- as received on the wire
   IN   wholeMsgLength             -- length as received on the wire
   IN   tmStateReference           -- (NEW) from the Transport Model
   OUT  securityEngineID           -- authoritative SNMP entity
   OUT  securityName               -- identification of the principal
   OUT  scopedPDU,                 -- message (plaintext) payload
   OUT  maxSizeResponseScopedPDU   -- maximum size sender can handle
   OUT  securityStateReference     -- reference to security state
                                   -- information, needed for response
   )
```

7. Security Considerations

This document defines an architectural approach that permits SNMP to utilize transport-layer security services. Each proposed Transport Model should discuss the security considerations of that Transport Model.

It is considered desirable by some industry segments that SNMP Transport Models utilize transport-layer security that addresses perfect forward secrecy at least for encryption keys. Perfect forward secrecy guarantees that compromise of long-term secret keys does not result in disclosure of past session keys. Each proposed Transport Model should include a discussion in its security considerations of whether perfect forward secrecy is appropriate for that Transport Model.

The denial-of-service characteristics of various Transport Models and security protocols will vary and should be evaluated when determining the applicability of a Transport Model to a particular deployment situation.

Since the cache will contain security-related parameters, implementers SHOULD store this information (in memory or in persistent storage) in a manner to protect it from unauthorized disclosure and/or modification.

Care must be taken to ensure that an SNMP engine is sending packets out over a transport using credentials that are legal for that engine to use on behalf of that user. Otherwise, an engine that has multiple transports open might be "tricked" into sending a message through the wrong transport.

A Security Model might have multiple sources from which to define the securityName and securityLevel. The use of a secure Transport Model does not imply that the securityName and securityLevel chosen by the Security Model represent the transport-authenticated identity or the transport-provided security services. The securityModel, securityName, and securityLevel parameters are a related set, and an administrator should understand how the specified securityModel selects the corresponding securityName and securityLevel.

7.1. Coexistence, Security Parameters, and Access Control

In the RFC 3411 architecture, the Message Processing Model makes the decision about which Security Model to use. The architectural change described by this document does not alter that.

The architecture change described by this document does, however, allow SNMP to support two different approaches to security -- message-driven security and transport-driven security. With message-driven security, SNMP provides its own security and passes security parameters within the SNMP message; with transport-driven security, SNMP depends on an external entity to provide security during transport by "wrapping" the SNMP message.

Using a non-transport-aware Security Model with a secure Transport Model is NOT RECOMMENDED for the following reasons.

Security Models defined before the Transport Security Model (i.e., SNMPv1, SNMPv2c, and USM) do not support transport-based security and only have access to the security parameters contained within the SNMP message. They do not know about the security parameters associated with a secure transport. As a result, the Access Control Subsystem bases its decisions on the security parameters extracted from the SNMP message, not on transport-based security parameters.

Implications of combining older Security Models with Secure Transport Models are known. The securityName used for access control decisions is based on the message-driven identity, which might be unauthenticated, and not on the transport-driven, authenticated identity:

- An SNMPv1 message will always be paired with an SNMPv1 Security Model (per RFC 3584), regardless of the transport mapping or Transport Model used, and access controls will be based on the unauthenticated community name.

- An SNMPv2c message will always be paired with an SNMPv2c Security Model (per RFC 3584), regardless of the transport mapping or Transport Model used, and access controls will be based on the unauthenticated community name.

- An SNMPv3 message will always be paired with the securityModel specified in the msgSecurityParameters field of the message (per RFC 3412), regardless of the transport mapping or Transport Model used. If the SNMPv3 message specifies the User-based Security Model (USM) with noAuthNoPriv, then the access controls will be based on the unauthenticated USM user.

- For outgoing messages, if a Secure Transport Model is selected in combination with a Security Model that does not populate a tmStateReference, the Secure Transport Model SHOULD detect the lack of a valid tmStateReference and fail.

In times of network stress, a Secure Transport Model might not work properly if its underlying security mechanisms (e.g., Network Time Protocol (NTP) or Authentication, Authorization, and Accounting (AAA) protocols or certificate authorities) are not reachable. The User-based Security Model was explicitly designed to not depend upon external network services, and provides its own security services. It is RECOMMENDED that operators provision authPriv USM as a fallback mechanism to supplement any Security Model or Transport Model that has external dependencies, so that secure SNMP communications can continue when the external network service is not available.

8. IANA Considerations

IANA has created a new registry in the Simple Network Management Protocol (SNMP) Number Spaces. The new registry is called "SNMP Transport Domains". This registry contains US-ASCII alpha-numeric strings of one to four characters to identify prefixes for corresponding SNMP transport domains. Each transport domain MUST have an OID assignment under snmpDomains [RFC2578]. Values are to be assigned via [RFC5226] "Specification Required".

The registry has been populated with the following initial entries:

Registry Name: SNMP Transport Domains
Reference: [RFC2578] [RFC3417] [RFC5590]
Registration Procedures: Specification Required
Each domain is assigned a MIB-defined OID under snmpDomains

Prefix	snmpDomains	Reference
udp	snmpUDPDomain	[RFC3417] [RFC5590]
clns	snmpCLNSDomain	[RFC3417] [RFC5590]
cons	snmpCONSDomain	[RFC3417] [RFC5590]
ddp	snmpDDPDomain	[RFC3417] [RFC5590]
ipx	snmpIPXDomain	[RFC3417] [RFC5590]
prxy	rfc1157Domain	[RFC3417] [RFC5590]

9. Acknowledgments

The Integrated Security for SNMP WG would like to thank the following people for their contributions to the process.

The authors of submitted Security Model proposals: Chris Elliot, Wes Hardaker, David Harrington, Keith McCloghrie, Kaushik Narayan, David Perkins, Joseph Salowey, and Juergen Schoenwaelder.

The members of the Protocol Evaluation Team: Uri Blumenthal, Lakshminath Dondeti, Randy Presuhn, and Eric Rescorla.

WG members who performed detailed reviews: Wes Hardaker, Jeffrey Hutzelman, Tom Petch, Dave Shield, and Bert Wijnen.

10. References

10.1. Normative References

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

[RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

[RFC3411] Harrington, D., Presuhn, R., and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, December 2002.

[RFC3412] Case, J., Harrington, D., Presuhn, R., and B. Wijnen, "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3412,

December 2002.

[RFC3413] Levi, D., Meyer, P., and B. Stewart, "Simple Network Management Protocol (SNMP) Applications", STD 62, RFC 3413, December 2002.

[RFC3414] Blumenthal, U. and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", STD 62, RFC 3414, December 2002.

[RFC3417] Presuhn, R., "Transport Mappings for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3417, December 2002.

10.2. Informative References

[RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.

[RFC3410] Case, J., Mundy, R., Partain, D., and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, December 2002.

[RFC3584] Frye, R., Levi, D., Routhier, S., and B. Wijnen, "Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework", BCP 74, RFC 3584, August 2003.

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

[RFC4422] Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.

[RFC4741] Enns, R., "NETCONF Configuration Protocol", RFC 4741, December 2006.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5424] Gerhards, R., "The Syslog Protocol", RFC 5424, March 2009.

[RFC5591] Harrington, D. and W. Hardaker, "Transport Security Model for the Simple Network Management Protocol (SNMP)", RFC 5591, June 2009.

[RFC5592] Harrington, D., Salowey, J., and W. Hardaker, "Secure Shell Transport Model for the Simple Network Management Protocol (SNMP)", RFC 5592, June 2009.

Appendix A. Why tmStateReference?

This appendix considers why a cache-based approach was selected for passing parameters.

There are four approaches that could be used for passing information between the Transport Model and a Security Model.

1. One could define an ASI to supplement the existing ASIs.

2. One could add a header to encapsulate the SNMP message.

3. One could utilize fields already defined in the existing SNMPv3 message.

4. One could pass the information in an implementation-specific cache or via a MIB module.

A.1. Define an Abstract Service Interface

Abstract Service Interfaces (ASIs) are defined by a set of primitives that specify the services provided and the abstract data elements that are to be passed when the services are invoked. Defining additional ASIs to pass the security and transport information from the Transport Subsystem to the Security Subsystem has the advantage of being consistent with existing RFC 3411/3412 practice; it also helps to ensure that any Transport Model proposals pass the necessary data and do not cause side effects by creating model-specific dependencies between itself and models or subsystems other than those that are clearly defined by an ASI.

A.2. Using an Encapsulating Header

A header could encapsulate the SNMP message to pass necessary information from the Transport Model to the Dispatcher and then to a Message Processing Model. The message header would be included in the wholeMessage ASI parameter and would be removed by a corresponding Message Processing Model. This would imply the (one and only) Message Dispatcher would need to be modified to determine which SNMP message version was involved, and a new Message Processing Model would need to be developed that knew how to extract the header from the message and pass it to the Security Model.

A.3. Modifying Existing Fields in an SNMP Message

[RFC3412] defines the SNMPv3 message, which contains fields to pass security-related parameters. The Transport Subsystem could use these fields in an SNMPv3 message (or comparable fields in other message

formats) to pass information between Transport Models in different SNMP engines and to pass information between a Transport Model and a corresponding Message Processing Model.

If the fields in an incoming SNMPv3 message are changed by the Transport Model before passing it to the Security Model, then the Transport Model will need to decode the ASN.1 message, modify the fields, and re-encode the message in ASN.1 before passing the message on to the Message Dispatcher or to the transport layer. This would require an intimate knowledge of the message format and message versions in order for the Transport Model to know which fields could be modified. This would seriously violate the modularity of the architecture.

A.4. Using a Cache

This document describes a cache into which the Transport Model (TM) puts information about the security applied to an incoming message; a Security Model can extract that information from the cache. Given that there might be multiple TM security caches, a tmStateReference is passed as an extra parameter in the ASIs between the Transport Subsystem and the Security Subsystem so that the Security Model knows which cache of information to consult.

This approach does create dependencies between a specific Transport Model and a corresponding specific Security Model. However, the approach of passing a model-independent reference to a model-dependent cache is consistent with the securityStateReference already being passed around in the RFC 3411 ASIs.

Authors' Addresses

David Harrington
Huawei Technologies (USA)
1700 Alma Dr. Suite 100
Plano, TX 75075
USA

Phone: +1 603 436 8634
EMail: ietfdbh@comcast.net

Juergen Schoenwaelder
Jacobs University Bremen
Campus Ring 1
28725 Bremen
Germany

Phone: +49 421 200-3587
EMail: j.schoenwaelder@jacobs-university.de