Network Working Group
Request for Comments: 5592
Category: Standards Track

D. Harrington
Huawei Technologies (USA)
J. Salowey
Cisco Systems
W. Hardaker
Cobham Analytic Solutions

June 2009

Secure Shell Transport Model for the Simple Network Management Protocol (SNMP)

Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Copyright (c) 2009 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents in effect on the date of publication of this document (http://trustee.ietf.org/license-info). Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Abstract

This memo describes a Transport Model for the Simple Network Management Protocol (SNMP), using the Secure Shell (SSH) protocol.

This memo also defines a portion of the Management Information Base (MIB) for use with network management protocols in TCP/IP-based internets. In particular, it defines objects for monitoring and managing the Secure Shell Transport Model for SNMP.

Table of Contents

1. Introduction

This memo describes a Transport Model for the Simple Network Management Protocol, using the Secure Shell (SSH) protocol [RFC4251] within a Transport Subsystem [RFC5590]. The Transport Model specified in this memo is referred to as the Secure Shell Transport Model (SSHTM).

This memo also defines a portion of the Management Information Base (MIB) for use with network management protocols in TCP/IP-based internets. In particular, it defines objects for monitoring and managing the Secure Shell Transport Model for SNMP.

It is important to understand the SNMP architecture [RFC3411] and the terminology of the architecture to understand where the Transport Model described in this memo fits into the architecture and interacts with other subsystems within the architecture.

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].

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. MIB objects are generally accessed through the Simple Network Management Protocol (SNMP). Objects in the MIB are defined using the mechanisms defined in the Structure of Management Information (SMI). This memo specifies a MIB module that is compliant to the SMIv2, which is described in STD 58, RFC 2578 [RFC2578], STD 58, RFC 2579 [RFC2579] and STD 58, RFC 2580 [RFC2580].

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 [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. 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.

"Authentication" in this document typically refers to the English meaning of "serving to prove the authenticity of" the message, not data source authentication or peer identity authentication.

The terms "manager" and "agent" are not used in this document because, in the RFC 3411 architecture, all SNMP entities have the capability of acting as manager, agent, or both depending on the SNMP application types supported in the implementation. Where distinction is required, the application names of command generator, command responder, notification originator, notification receiver, and proxy forwarder are used. See "SNMP Applications" [RFC3413] for further information.

The User-based Security Model (USM) [RFC3414] is a mandatory-to-implement Security Model in STD 62. While the SSH and USM specifications frequently refer to a user, the terminology preferred in [RFC3411] and in this memo is "principal". A principal is the "who" on whose behalf services are provided or processing takes place. A principal can be, among other things, an individual acting in a particular role, a set of individuals each acting in a particular role, an application or a set of applications, or a combination of these within an administrative domain.

Throughout this document, the terms "client" and "server" are used to refer to the two ends of the SSH transport connection. The client actively opens the SSH connection, and the server passively listens for the incoming SSH connection. Either SNMP entity may act as client or as server, as discussed further below.

1.3. Modularity

The reader is expected to have read and understood the description of the SNMP architecture, as defined in [RFC3411], and the Transport Subsystem architecture extension specified in "Transport Subsystem for the Simple Network Management Protocol (SNMP)" [RFC5590].

This memo describes the Secure Shell Transport Model for SNMP, a specific SNMP Transport Model to be used within the SNMP Transport Subsystem to provide authentication, encryption, and integrity checking of SNMP messages.

In keeping with the RFC 3411 design decision to use self-contained documents, this document defines the elements of procedure and associated MIB module objects that are needed for processing the Secure Shell Transport Model for SNMP.

This modularity of specification is not meant to be interpreted as imposing any specific requirements on implementation.

1.4. Motivation

Version 3 of the Simple Network Management Protocol (SNMPv3) added security to the protocol. The User-based Security Model (USM) [RFC3414] was designed to be independent of other existing security infrastructures to ensure it could function when third-party authentication services were not available, such as in a broken network. As a result, USM utilizes a separate user and key-management infrastructure. Operators have reported that having to deploy another user and key-management infrastructure in order to use SNMPv3 is a reason for not deploying SNMPv3.

This memo describes a Transport Model that will make use of the existing and commonly deployed Secure Shell security infrastructure. This Transport Model is designed to meet the security and operational needs of network administrators, maximize usability in operational environments to achieve high deployment success, and at the same time minimize implementation and deployment costs to minimize deployment time.

This document addresses the requirement for the SSH client to authenticate the SSH server and for the SSH server to authenticate the SSH client, and describes how SNMP can make use of the authenticated identities in authorization policies for data access, in a manner that is independent of any specific Access Control Model.

This document addresses the requirement to utilize client-authentication and key-exchange methods that support different security infrastructures and provide different security properties. This document describes how to use client authentication as described in "The Secure Shell (SSH) Authentication Protocol" [RFC4252]. The SSH Transport Model should work with any of the ssh-userauth methods, including the "publickey", "password", "hostbased", "none", "keyboard-interactive", "gssapi-with-mic", ."gssapi-keyex", "gssapi", and "external-keyx" (see the SSH Protocol Parameters registry maintained by IANA). The use of the "none" authentication method is NOT RECOMMENDED, as described in this document's Security Considerations. Local accounts may be supported through the use of the publickey, hostbased, or password methods. The password method allows for integration with a deployed password infrastructure, such as Authentication, Authorization, and Accounting (AAA) servers using the RADIUS protocol [RFC2865]. The SSH Transport Model SHOULD be able to take advantage of future-defined ssh-userauth methods, such as those that might make use of X.509 certificate credentials.

It is desirable to use mechanisms that could unify the approach for administrative security for SNMPv3 and command line interfaces (CLI) and other management interfaces. The use of security services provided by Secure Shell is the approach commonly used for the CLI and is the approach being adopted for use with NETCONF [RFC4742]. This memo describes a method for invoking and running the SNMP protocol within a Secure Shell (SSH) session as an SSH Subsystem.

This memo describes how SNMP can be used within a Secure Shell (SSH) session, using the SSH connection protocol [RFC4254] over the SSH transport protocol, and using ssh-userauth [RFC4252] for authentication.

There are a number of challenges to be addressed to map Secure Shell authentication method parameters into the SNMP architecture so that SNMP continues to work without any surprises. These are discussed in detail below.

1.5. Constraints

The design of this SNMP Transport Model is influenced by the following constraints:

1. In times of network stress, the transport protocol and its underlying security mechanisms SHOULD NOT depend upon the ready availability of other network services (e.g., Network Time Protocol (NTP) or AAA protocols).

2. When the network is not under stress, the Transport Model and its underlying security mechanisms MAY depend upon the ready availability of other network services.

3. It may not be possible for the Transport Model to determine when the network is under stress.

4. A Transport Model SHOULD NOT require changes to the SNMP architecture.

5. A Transport Model SHOULD NOT require changes to the underlying security protocol.

2. The Secure Shell Protocol

SSH is a protocol for secure remote login and other secure network services over an insecure network. It consists of three major protocol components and add-on methods for user authentication:

- The Transport Layer Protocol [RFC4253] provides server authentication and message confidentiality and integrity. It may optionally also provide compression. The transport layer will typically be run over a TCP/IP connection but might also be used on top of any other reliable data stream.

- The User Authentication Protocol [RFC4252] authenticates the client-side principal to the server. It runs over the Transport Layer Protocol.

- The Connection Protocol [RFC4254] multiplexes the encrypted tunnel into several logical channels. It runs over the transport after successfully authenticating the principal.

- Generic Message Exchange Authentication [RFC4256] is a general purpose authentication method for the SSH protocol, suitable for interactive authentications where the authentication data should be entered via a keyboard.

- "Generic Security Service Application Program Interface (GSS-API) Authentication and Key Exchange for the Secure Shell (SSH) Protocol" [RFC4462] describes methods for using the GSS-API for authentication and key exchange in SSH. It defines an SSH user-authentication method that uses a specified GSS-API mechanism to authenticate a user; it also defines a family of SSH key-exchange methods that use GSS-API to authenticate a Diffie-Hellman key exchange.

The client sends a service request once a secure, transport-layer connection has been established. A second service request is sent after client authentication is complete. This allows new protocols to be defined and coexist with the protocols listed above.

The connection protocol provides channels that can be used for a wide range of purposes. Standard methods are provided for setting up secure interactive shell sessions and for forwarding ("tunneling") arbitrary TCP/IP ports and X11 connections.

3. How SSHTM Fits into the Transport Subsystem

A Transport Model is a component of the Transport Subsystem [RFC5590] within the SNMP architecture. The SSH Transport Model thus fits between the underlying SSH transport layer and the Message Dispatcher [RFC3411].

The SSH Transport Model will establish a channel between itself and the SSH Transport Model of another SNMP engine. The sending Transport Model passes unencrypted messages from the Dispatcher to SSH to be encrypted, and the receiving Transport Model accepts decrypted incoming messages from SSH and passes them to the Dispatcher.

After an SSH Transport Model channel is established, then SNMP messages can conceptually be sent through the channel from one SNMP Message Dispatcher to another SNMP Message Dispatcher. Multiple SNMP messages MAY be passed through the same channel.

The SSH Transport Model of an SNMP engine will perform the translation between SSH-specific security parameters and SNMP-specific, model-independent parameters.

3.1. Security Capabilities of this Model

3.1.1. Threats

The Secure Shell Transport Model provides protection against the threats identified by the RFC 3411 architecture [RFC3411]:

1. Modification of Information - SSH provides for verification that the contents of each message have not been modified during its transmission through the network by digitally signing each SSH packet.

2. Masquerade - SSH provides for verification of the identity of the SSH server and the identity of the SSH client.

SSH provides for verification of the identity of the SSH server through the SSH transport protocol server authentication [RFC4253]. This allows an operator or management station to ensure the authenticity of the SNMP engine that provides MIB data.

SSH provides a number of mechanisms for verification of the identity of the SSH client-side principal using the Secure Shell Authentication Protocol [RFC4252]. These include public key, password, and host-based mechanisms. This allows the SNMP Access Control Subsystem to ensure that only authorized principals have access to potentially sensitive data.

Verification of the client's principal identity is important for use with the SNMP Access Control Subsystem to ensure that only authorized principals have access to potentially sensitive data.

The SSH user identity is provided to the Transport Model, so it can be used to map to an SNMP model-independent securityName for use with SNMP access control and notification configuration. (The identity may undergo various transforms before it maps to the securityName.)

3. Message Stream Modification - SSH protects against malicious re-ordering or replaying of messages within a single SSH session by using sequence numbers and integrity checks. SSH protects against replay of messages across SSH sessions by ensuring that the cryptographic keys used for encryption and integrity checks are generated afresh for each session.

4. Disclosure - SSH provides protection against the disclosure of information to unauthorized recipients or eavesdroppers by allowing for encryption of all traffic between SNMP engines.

3.1.2. Message Authentication

The RFC 3411 architecture recognizes three levels of security:

- without authentication and without privacy (noAuthNoPriv)

- with authentication but without privacy (authNoPriv)

- with authentication and with privacy (authPriv)

The Secure Shell protocol provides support for encryption and data integrity. While it is technically possible to support no authentication and no encryption in SSH, it is NOT RECOMMENDED by [RFC4253].

The SSH Transport Model determines from SSH the identity of the authenticated principal and the type and address associated with an incoming message, and provides this information to SSH for an outgoing message. The SSH transport-layer algorithms used to provide authentication, data integrity, and encryption SHOULD NOT be exposed to the SSH Transport Model layer. The SNMPv3 WG deliberately avoided this and settled for an assertion by the Security Model that the requirements of securityLevel were met. The SSH Transport Model has no mechanisms by which it can test whether an underlying SSH connection provides auth or priv, so the SSH Transport Model trusts that the underlying SSH connection has been properly configured to support authPriv security characteristics.

An SSH Transport-Model-compliant implementation MUST use an SSH connection that provides authentication, data integrity, and encryption that meets the highest level of SNMP security (authPriv). Outgoing messages specified with a securityLevel of noAuthNoPriv or authNoPriv are actually sent by the SSH Transport Model with authPriv-level protection.

The security protocols used in the Secure Shell Authentication Protocol [RFC4252] and the Secure Shell Transport Layer Protocol [RFC4253] are considered acceptably secure at the time of writing. However, the procedures allow for new authentication and privacy methods to be specified at a future time if the need arises.

3.1.3. Authentication Protocol Support

The SSH Transport Model should support any server- or client-authentication mechanism supported by SSH. This includes the three authentication methods described in the SSH Authentication Protocol document [RFC4252] (publickey, password, and host-based), keyboard interactive, and others.

The password-authentication mechanism allows for integration with deployed password-based infrastructure. It is possible to hand a password to a service such as RADIUS [RFC2865] or Diameter [RFC3588] for validation. The validation could be done using the user name and user password attributes. It is also possible to use a different password-validation protocol such as the Challenge Handshake Authentication Protocol (CHAP) [RFC1994] or digest authentication [RFC5090] to integrate with RADIUS or Diameter. At some point in the processing, these mechanisms require the password to be made available as cleartext on the device that is authenticating the password, which might introduce threats to the authentication infrastructure.

GSS-API key exchange [RFC4462] provides a framework for the addition of client-authentication mechanisms that support different security infrastructures and provide different security properties. Additional authentication mechanisms, such as one that supports X.509 certificates, may be added to SSH in the future.

3.1.4. SSH Subsystem

This document describes the use of an SSH Subsystem for SNMP to make SNMP usage distinct from other usages.

An SSH Subsystem of type "snmp" is opened by the SSH Transport Model during the elements of procedure for an outgoing SNMP message. Since the sender of a message initiates the creation of an SSH session if needed, the SSH session will already exist for an incoming message; otherwise, the incoming message would never reach the SSH Transport Model.

Implementations may choose to instantiate SSH sessions in anticipation of outgoing messages. This approach might be useful to ensure that an SSH session to a given target can be established before it becomes important to send a message over the SSH session. Of course, there is no guarantee that a pre-established session will still be valid when needed.

SSH sessions are uniquely identified within the SSH Transport Model by the combination of tmTransportAddress and tmSecurityName associated with each session.

Because naming policies might differ between administrative domains, many SSH client software packages support a user@hostname:port addressing syntax that operators can use to align non-equivalent account names. The SnmpSSHAddress Textual Convention echos this common SSH notation.

When this notation is used in an SnmpSSHAddress, the SSH connection should be established with an SSH user name matching the "user" portion of the notation when establishing a session with the remote SSH server. The user name must be encoded in UTF-8 (per [RFC4252]). The "user" portion may or may not match the tmSecurityName parameter passed from the Security Model. If no "user@" portion is specified in the SnmpSSHAddress, then the SSH connection should be established using the tmSecurityName as the SSH user name when establishing a session with the remote SSH server.

The SnmpSSHAddress and tmSecurityName associated with an SSH session MUST remain constant during the life of the session. Different SnmpSSHAddress values (with different hostnames, "user@" prefix names, and/or port numbers) will each result in individual SSH sessions.

3.2. Security Parameter Passing

For incoming messages, SSH-specific security parameters are translated by the Transport Model into security parameters independent of the Transport and Security Models. The Transport Model accepts messages from the SSH Subsystem, records the transport-related and SSH-security-related information, including the authenticated identity, in a cache referenced by tmStateReference, and passes the WholeMsg and the tmStateReference to the Dispatcher using the receiveMessage() ASI (Abstract Service Interface).

For outgoing messages, the Transport Model takes input provided by the Dispatcher in the sendMessage() ASI. The SSH Transport Model converts that information into suitable security parameters for SSH, establishes sessions as needed, and passes messages to the SSH Subsystem for sending.

3.3. Notifications and Proxy

SSH connections may be initiated by command generators or by notification originators. Command generators are frequently operated by a human, but notification originators are usually unmanned automated processes. As a result, it may be necessary to provision authentication credentials on the SNMP engine containing the notification originator or to use a third-party key provider, such as Kerberos, so the engine can successfully authenticate to an engine containing a notification receiver.

The targets to whom notifications or proxy requests should be sent is typically determined and configured by a network administrator. The SNMP-NOTIFICATION-MIB contains a list of targets to which notifications should be sent. The SNMP-TARGET-MIB module [RFC3413] contains objects for defining these management targets, including transport domains and addresses and security parameters, for applications such as notification generators and proxy forwarders.

For the SSH Transport Model, transport type and address are configured in the snmpTargetAddrTable, and the securityName and securityLevel parameters are configured in the snmpTargetParamsTable. The default approach is for an administrator to statically preconfigure this information to identify the targets authorized to receive notifications or received proxied messages. Local access-

control processing needs to be performed by a notification originator before notifications are actually sent, and this processing is done using the configured securityName. An important characteristic of this is that authorization is done prior to determining if the connection can succeed. Thus, the locally configured securityName is entirely trusted within the notification originator.

The SNMP-TARGET-MIB and NOTIFICATION-MIB MIB modules may be configured using SNMP or other implementation-dependent mechanisms, such as CLI scripting or loading a configuration file. It may be necessary to provide additional implementation-specific configuration of SSH parameters.

4. Cached Information and References

When performing SNMP processing, there are two levels of state information that may need to be retained: the immediate state linking a request-response pair and a potentially longer-term state relating to transport and security. "Transport Subsystem for the Simple Network Management Protocol" [RFC5590] defines general requirements for caches and references.

This document defines additional cache requirements related to the Secure Shell Transport Model.

4.1. Secure Shell Transport Model Cached Information

The Secure Shell Transport Model has specific responsibilities regarding the cached information. See the Elements of Procedure in Section 5 for detailed processing instructions on the use of the tmStateReference fields by the SSH Transport Model.

4.1.1. tmSecurityName

The tmSecurityName MUST be a human-readable name (in snmpAdminString format) representing the identity that has been set according to the procedures in Section 5. The tmSecurityName MUST be constant for all traffic passing through an SSHTM session. Messages MUST NOT be sent through an existing SSH session that was established using a different tmSecurityName.

On the SSH server side of a connection:

- The tmSecurityName should be the SSH user name. How the SSH user name is extracted from the SSH layer is implementation-dependent.

The SSH protocol is not always clear on whether the user name field must be filled in, so for some implementations, such as those using GSSAPI authentication, it may be necessary to use a mapping algorithm to transform an SSH identity to a tmSecurityName or to transform a tmSecurityName to an SSH identity.

In other cases, the user name may not be verified by the server, so for these implementations, it may be necessary to obtain the user name from other credentials exchanged during the SSH exchange.

On the SSH client side of a connection:

The tmSecurityName is presented to the SSH Transport Model by the application (possibly because of configuration specified in the SNMP-TARGET-MIB).

The securityName MAY be derived from the tmSecurityName by a Security Model and MAY be used to configure notifications and access controls in MIB modules. Transport Models SHOULD generate a predictable tmSecurityName so operators will know what to use when configuring MIB modules that use securityNames derived from tmSecurityNames.

4.1.2. tmSessionID

The tmSessionID MUST be recorded per message at the time of receipt. When tmSameSecurity is set, the recorded tmSessionID can be used to determine whether the SSH session available for sending a corresponding outgoing message is the same SSH session as was used when receiving the incoming message (e.g., a response to a request).

4.1.3. Session State

The per-session state that is referenced by tmStateReference may be saved across multiple messages in a Local Configuration Datastore. Additional session/connection state information might also be stored in a Local Configuration Datastore.

5. Elements of Procedure

Abstract Service Interfaces have been defined by [RFC3411] and further augmented by [RFC5590] to describe the conceptual data flows between the various subsystems within an SNMP entity. The Secure Shell Transport Model uses some of these conceptual data flows when communicating 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 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.

An error indication in statusInformation will typically include the Object Identifier (OID) and value for an incremented error counter. This may be accompanied by the requested securityLevel and the tmStateReference. Per-message context information is not accessible to Transport Models, so for the returned counter OID and value, contextEngine would be set to the local value of snmpEngineID and contextName to the default context for error counters.

5.1. Procedures for an Incoming Message

1. The SSH Transport Model queries the SSH engine, in an implementation-dependent manner, to determine the address the message originated from, the user name authenticated by SSH, and a session identifier.

2. Determine the tmTransportAddress to be associated with the incoming message:

   A. If this is a client-side SSH session, then the tmTransportAddress is set to the tmTransportAddress used to establish the session. It MUST exactly include any "user@" prefix associated with the address provided to the openSession() ASI.

   B. If this is a server-side SSH session and this is the first message received over the session, then the tmTransportAddress is set to the address the message originated from, determined in an implementation-dependent way. This value MUST be constant for the entire SSH session, and future messages received MUST result in the tmTransportAddress being set to the same value.

   C. If this is a server-side SSH session and this is not the first message received over the session, then the tmTransportAddress is set to the previously established tmTransportAddress for the session (the value from step B, determined from a previous incoming message).

3. Determine the tmSecurityName to be associated with the incoming message:

   A. If this is a client-side SSH session, then the tmSecurityName MUST be set to the tmSecurityName used to establish the session.

   B. If this is a server-side SSH session and this is the first message received over the session, then the tmSecurityName is set to the SSH user name. How the SSH user name is extracted from the SSH layer is implementation-dependent. This value MUST be constant for the entire SSH session, and future messages received MUST result in the tmSecurityName being set to the same value.

   C. If this is a server-side SSH session and this is not the first message received over the session, then the tmSecurityName is set to the previously established tmSecurityName for the session (the value from step B, determined from a previous incoming message).

4. Create a tmStateReference cache for subsequent reference to the information.

   tmTransportDomain = snmpSSHDomain

   tmTransportAddress = the derived tmTransportAddress from step 2.

   tmSecurityName = the derived tmSecurityName from step 3.

   tmTransportSecurityLevel = "authPriv" (authentication and confidentiality MUST be used to comply with this Transport Model.)

   tmSessionID = an implementation-dependent value that can be used to detect when a session has closed and been replaced by another session. The value in tmStateReference MUST uniquely identify the session over which the message was received. This session identifier MUST NOT be reused until there are no references to it remaining.

Then the Transport Model passes the message to the Dispatcher using the following ASI:

```
statusInformation =
receiveMessage(
IN   transportDomain       -- snmpSSHDomain
IN   transportAddress      -- the tmTransportAddress for the message
IN   wholeMessage          -- the whole SNMP message from SSH
IN   wholeMessageLength    -- the length of the SNMP message
IN   tmStateReference      -- (NEW) transport info
 )
```

5.2. Procedures for Sending an Outgoing Message

The Dispatcher passes the information to the Transport Model using the ASI defined in the Transport Subsystem:

```
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              -- (NEW) transport info
)
```

The SSH Transport Model performs the following tasks.

1. If tmStateReference does not refer to a cache containing values for tmTransportDomain, tmTransportAddress, tmSecurityName, tmRequestedSecurityLevel, and tmSameSecurity, then increment the snmpSshtmSessionInvalidCaches counter, discard the message, and return the error indication in the statusInformation. Processing of this message stops.

2. Extract the tmTransportDomain, tmTransportAddress, tmSecurityName, tmRequestedSecurityLevel, tmSameSecurity, and tmSessionID from the tmStateReference.

3. Identify an SSH session over which to send the messages:

   A. If tmSameSecurity is true and there is no existing session with a matching tmSessionID, tmSecurityName, and tmTransportAddress, then increment the snmpSshtmSessionNoSessions counter, discard the message, and return the error indication in the statusInformation. Processing of this message stops.

   B. If there is a session with a matching tmSessionID, tmTransportAddress, and tmSecurityName, then select that session.

C. If there is a session that matches the tmTransportAddress and tmSecurityName, then select that session.

D. If the above steps failed to select a session to use, then call openSession() with the tmStateReference as a parameter.

+ If openSession fails, then discard the message, release tmStateReference, and pass the error indication returned by openSession back to the calling module. Processing of this message stops.

+ If openSession succeeds, then record the destTransportDomain, destTransportAddress, tmSecurityname, and tmSessionID in an implementation-dependent manner. This will be needed when processing an incoming message.

4. Pass the wholeMessage to SSH for encapsulation as data in an SSH message over the identified SSH session. Any necessary additional SSH-specific parameters should be provided in an implementation-dependent manner.

5.3. Establishing a Session

The Secure Shell Transport Model provides the following Abstract Service Interface (ASI) to describe the data passed between the SSH Transport Model and the SSH service. It is an implementation decision how such data is passed.

```
statusInformation =
openSession(
IN   tmStateReference       -- transport information to be used
OUT  tmStateReference       -- transport information to be used
IN   maxMessageSize         -- of the sending SNMP entity
 )
```

The following describes the procedure to follow to establish a session between a client and server to run SNMP over SSH. This process is used by any SNMP engine establishing a session for subsequent use.

This will be done automatically for an SNMP application that initiates a transaction, such as a command generator, a notification originator, or a proxy forwarder.

1. Increment the snmpSshtmSessionOpens counter.

2. Using tmTransportAddress, the client will establish an SSH transport connection using the SSH transport protocol, authenticate the server, and exchange keys for message integrity and encryption. The transportAddress associated with a session MUST remain constant during the lifetime of the SSH session. Implementations may need to cache the transportAddress passed to the openSession API for later use when performing incoming message processing (see Section 5.1).

    1. To authenticate the server, the client usually stores pairs (tmTransportAddress, server host public key) in an implementation-dependent manner.

    2. The other parameters of the transport connection are provided in an implementation-dependent manner.

    3. If the attempt to establish a connection is unsuccessful or if server-authentication fails, then snmpSshtmSessionOpenErrors is incremented, an openSession error indication is returned, and openSession processing stops.

3. The client will then invoke an SSH authentication service to authenticate the principal, such as that described in the SSH authentication protocol [RFC4252].

    1. If the tmTransportAddress field contains a user name followed by an '@' character (US-ASCII 0x40), that user name string should be presented to the SSH server as the "user name" for user-authentication purposes. If there is no user name in the tmTransportAddress, then the tmSecurityName should be used as the user name.

    2. The credentials used to authenticate the SSH principal are determined in an implementation-dependent manner.

    3. In an implementation-specific manner, invoke the SSH user-authentication service using the calculated user name.

    4. If the user authentication is unsuccessful, then the transport connection is closed, the snmpSshtmSessionUserAuthFailures counter is incremented, an error indication is returned to the calling module, and processing stops for this message.

4. The client should invoke the "ssh-connection" service (also known as the SSH connection protocol [RFC4254]), and request a channel of type "session". If unsuccessful, the transport connection is closed, the snmpSshtmSessionNoChannels counter is incremented, an error indication is returned to the calling module, and processing stops for this message.

5. The client invokes "snmp" as an SSH Subsystem, as indicated in the "subsystem" parameter. If unsuccessful, the transport connection is closed, the snmpSshtmSessionNoSubsystems counter is incremented, an error indication is returned to the calling module, and processing stops for this message.

   In order to allow SNMP traffic to be easily identified and filtered by firewalls and other network devices, servers associated with SNMP entities using the Secure Shell Transport Model MUST default to providing access to the "snmp" SSH Subsystem if the SSH session is established using the IANA-assigned TCP ports (5161 and 5162). Servers SHOULD be configurable to allow access to the SNMP SSH Subsystem over other ports.

6. Set tmSessionID in the tmStateReference cache to an implementation-dependent value to identify the session.

7. The tmSecurityName used to establish the SSH session must be the only tmSecurityName used with the session. Incoming messages for the session MUST be associated with this tmSecurityName value. How this is accomplished is implementation-dependent.

5.4. Closing a Session

The Secure Shell Transport Model provides the following ASI to close a session:

```
statusInformation =
closeSession(
IN   tmSessionID     -- session ID of session to be closed
)
```

The following describes the procedure to follow to close a session between a client and server. This process is followed by any SNMP engine to close an SSH session. It is implementation-dependent when a session should be closed. The calling code should release the associated tmStateReference.

1. Increment the snmpSshtmSessionCloses counter.

2. If there is no session corresponding to tmSessionID, then closeSession processing is complete.

3. Have SSH close the session associated with tmSessionID.

6. MIB Module Overview

This MIB module provides management of the Secure Shell Transport Model. It defines an OID to identify the SNMP-over-SSH transport domain, a Textual Convention for SSH Addresses, and several statistics counters.

6.1. Structure of the MIB Module

Objects in this MIB module are arranged into subtrees. Each subtree is organized as a set of related objects. The overall structure and assignment of objects to their subtrees, and the intended purpose of each subtree, is shown below.

6.2. Textual Conventions

Generic and Common Textual Conventions used in this document can be found summarized at http://www.ops.ietf.org/mib-common-tcs.html

6.3. Relationship to Other MIB Modules

Some management objects defined in other MIB modules are applicable to an entity implementing the SSH Transport Model. In particular, it is assumed that an entity implementing the SNMP-SSH-TM-MIB will implement the SNMPv2-MIB [RFC3418] and the SNMP-FRAMEWORK-MIB [RFC3411]. It is expected that an entity implementing this MIB will also support the Transport Security Model [RFC5591] and, therefore, implement the SNMP-TSM-MIB.

This MIB module is for monitoring SSH Transport Model information.

6.3.1. MIB Modules Required for IMPORTS

The following MIB module imports items from [RFC2578], [RFC2579], and [RFC2580].

This MIB module also references [RFC1033], [RFC4252], [RFC3490], and [RFC3986].

This document uses TDomain Textual Conventions for the SNMP-internal MIB modules defined here for compatibility with the RFC 3413 MIB modules and the RFC 3411 Abstract Service Interfaces.

7. MIB Module Definition

```
SNMP-SSH-TM-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE,
    OBJECT-IDENTITY, mib-2, snmpDomains,
    Counter32
      FROM SNMPv2-SMI -- RFC 2578
    TEXTUAL-CONVENTION
      FROM SNMPv2-TC -- RFC 2579
    MODULE-COMPLIANCE, OBJECT-GROUP
      FROM SNMPv2-CONF -- RFC 2580
    ;

snmpSshtmMIB MODULE-IDENTITY
    LAST-UPDATED "200906090000Z"
    ORGANIZATION "ISMS Working Group"
    CONTACT-INFO "WG-EMail:   isms@lists.ietf.org
                  Subscribe:  isms-request@lists.ietf.org

                  Chairs:
                    Juergen Quittek
                    NEC Europe Ltd.
                    Network Laboratories
                    Kurfuersten-Anlage 36
                    69115 Heidelberg
                    Germany
                    +49 6221 90511-15
                    quittek@netlab.nec.de

                    Juergen Schoenwaelder
                    Jacobs University Bremen
                    Campus Ring 1
                    28725 Bremen
                    Germany
                    +49 421 200-3587
                    j.schoenwaelder@jacobs-university.de

                  Co-editors:
                    David Harrington
                    Huawei Technologies USA
                    1700 Alma Drive
                    Plano Texas 75075
```

```
                   USA
                   +1 603-436-8634
                   ietfdbh@comcast.net

                   Joseph Salowey
                   Cisco Systems
                   2901 3rd Ave
                   Seattle, WA 98121
                   USA
                   jsalowey@cisco.com

                   Wes Hardaker
                   Cobham Analytic Solutions
                   P.O. Box 382
                   Davis, CA  95617
                   USA
                   +1 530 792 1913
                   ietf@hardakers.net
                "
    DESCRIPTION
       "The Secure Shell Transport Model MIB.

        Copyright (c) 2009 IETF Trust and the persons
        identified as authors of the code.  All rights reserved.

        Redistribution and use in source and binary forms, with or
        without modification, are permitted provided that the
        following conditions are met:

        - Redistributions of source code must retain the above copyright
          notice, this list of conditions and the following disclaimer.

        - Redistributions in binary form must reproduce the above
          copyright notice, this list of conditions and the following
          disclaimer in the documentation and/or other materials
          provided with the distribution.

        - Neither the name of Internet Society, IETF or IETF Trust,
          nor the names of specific contributors, may be used to endorse
          or promote products derived from this software without
          specific prior written permission.

        THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
        CONTRIBUTORS 'AS IS' AND ANY EXPRESS OR IMPLIED WARRANTIES,
        INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
        MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
        DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
        CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
```

```
        SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
        NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
        LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
        HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
        CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
        OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
        EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

        This version of this MIB module is part of RFC 5592;
        see the RFC itself for full legal notices."

    REVISION     "200906090000Z"
    DESCRIPTION  "The initial version, published in RFC 5592."

    ::= { mib-2 189 }

-- ------------------------------------------------------------ --
-- subtrees in the SNMP-SSH-TM-MIB
-- ------------------------------------------------------------ --

snmpSshtmNotifications    OBJECT IDENTIFIER ::= { snmpSshtmMIB 0 }
snmpSshtmObjects          OBJECT IDENTIFIER ::= { snmpSshtmMIB 1 }
snmpSshtmConformance      OBJECT IDENTIFIER ::= { snmpSshtmMIB 2 }

-- ---------------------------------------------------------------
-- Objects
-- ---------------------------------------------------------------

snmpSSHDomain OBJECT-IDENTITY
    STATUS      current
    DESCRIPTION
        "The SNMP-over-SSH transport domain.  The corresponding
         transport address is of type SnmpSSHAddress.

         When an SNMP entity uses the snmpSSHDomain Transport
         Model, it must be capable of accepting messages up to
         and including 8192 octets in size.  Implementation of
         larger values is encouraged whenever possible.

         The securityName prefix to be associated with the
         snmpSSHDomain is 'ssh'.  This prefix may be used by Security
         Models or other components to identify which secure transport
         infrastructure authenticated a securityName."
    ::= { snmpDomains 7 }

SnmpSSHAddress ::= TEXTUAL-CONVENTION
    DISPLAY-HINT "1a"
    STATUS      current
```

```
    DESCRIPTION
        "Represents either a hostname or IP address, along with a port
         number and an optional user name.

         The beginning of the address specification may contain a
         user name followed by an '@' (US-ASCII character 0x40).  This
         portion of the address will indicate the user name that should
         be used when authenticating to an SSH server.  The user name
         must be encoded in UTF-8 (per [RFC4252]).  If missing, the
         SNMP securityName should be used.  After the optional user
         name field and '@' character comes the hostname or IP
         address.

         The hostname is always in US-ASCII (as per RFC1033);
         internationalized hostnames are encoded in US-ASCII as
         specified in RFC 3490.  The hostname is followed by a colon
         ':' (US-ASCII character 0x3A) and a decimal port number in
         US-ASCII.  The name SHOULD be fully qualified whenever
         possible.

         An IPv4 address must be in dotted decimal format followed
         by a colon ':' (US-ASCII character 0x3A) and a decimal port
         number in US-ASCII.

         An IPv6 address must be in colon-separated format, surrounded
         by square brackets ('[', US-ASCII character 0x5B, and ']',
         US-ASCII character 0x5D), followed by a colon ':' (US-ASCII
         character 0x3A) and a decimal port number in US-ASCII.

         Values of this Textual Convention might not be directly usable
         as transport-layer addressing information and may require
         runtime resolution.  As such, applications that write them
         must be prepared for handling errors if such values are
         not supported or cannot be resolved (if resolution occurs
         at the time of the management operation).

         The DESCRIPTION clause of TransportAddress objects that may
         have snmpSSHAddress values must fully describe how (and
         when) such names are to be resolved to IP addresses and vice
         versa.

         This Textual Convention SHOULD NOT be used directly in
         object definitions since it restricts addresses to a
         specific format.  However, if it is used, it MAY be used
         either on its own or in conjunction with
         TransportAddressType or TransportDomain as a pair.
```

```
          When this Textual Convention is used as a syntax of an
          index object, there may be issues with the limit of 128
          sub-identifiers, which is specified in SMIv2 (STD 58).  It
          is RECOMMENDED that all MIB documents using this Textual
          Convention make explicit any limitations on index
          component lengths that management software must observe.
          This may be done either by including SIZE constraints on
          the index components or by specifying applicable
          constraints in the conceptual row DESCRIPTION clause or
          in the surrounding documentation.
        "
    REFERENCE
      "RFC 1033: DOMAIN ADMINISTRATORS OPERATIONS GUIDE
       RFC 3490: Internationalizing Domain Names in Applications
       RFC 3986: Uniform Resource Identifier (URI): Generic Syntax
       RFC 4252: The Secure Shell (SSH) Authentication Protocol"
    SYNTAX      OCTET STRING (SIZE (1..255))

-- The snmpSshtmSession Group

snmpSshtmSession       OBJECT IDENTIFIER ::= { snmpSshtmObjects 1 }

snmpSshtmSessionOpens  OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "The number of times an openSession() request has been
                 executed as an SSH client, whether it succeeded or
                 failed.
                "
    ::= { snmpSshtmSession 1 }

snmpSshtmSessionCloses  OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "The number of times a closeSession() request has been
                 executed as an SSH client, whether it succeeded or
                 failed.
                "
    ::= { snmpSshtmSession 2 }

snmpSshtmSessionOpenErrors  OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
```

```
    DESCRIPTION "The number of times an openSession() request
                 failed to open a transport connection or failed to
                 authenticate the server.
                "
    ::= { snmpSshtmSession 3 }

snmpSshtmSessionUserAuthFailures  OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "The number of times an openSession() request
                 failed to open a session as an SSH client due to
                 user-authentication failures.
                "
    ::= { snmpSshtmSession 4 }

snmpSshtmSessionNoChannels  OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "The number of times an openSession() request
                 failed to open a session as an SSH client due to
                 channel-open failures.
                "
    ::= { snmpSshtmSession 5 }

snmpSshtmSessionNoSubsystems OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "The number of times an openSession() request
                 failed to open a session as an SSH client due to
                 inability to connect to the requested subsystem.
                "
    ::= { snmpSshtmSession 6 }

snmpSshtmSessionNoSessions  OBJECT-TYPE
    SYNTAX       Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "The number of times an outgoing message was
                 dropped because the same session was no longer
                 available.
                "
    ::= { snmpSshtmSession 7 }

snmpSshtmSessionInvalidCaches OBJECT-TYPE
    SYNTAX       Counter32
```

```
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION "The number of outgoing messages dropped because the
                 tmStateReference referred to an invalid cache.
                "
    ::= { snmpSshtmSession 8 }

-- ************************************************
-- snmpSshtmMIB - Conformance Information
-- ************************************************

snmpSshtmCompliances OBJECT IDENTIFIER ::= { snmpSshtmConformance 1 }

snmpSshtmGroups      OBJECT IDENTIFIER ::= { snmpSshtmConformance 2 }

-- ************************************************
-- Compliance statements
-- ************************************************

snmpSshtmCompliance MODULE-COMPLIANCE
    STATUS      current

    DESCRIPTION "The compliance statement for SNMP engines that
                 support the SNMP-SSH-TM-MIB."
    MODULE
        MANDATORY-GROUPS { snmpSshtmGroup }
    ::= { snmpSshtmCompliances 1 }

-- ************************************************
-- Units of conformance
-- ************************************************

snmpSshtmGroup OBJECT-GROUP
    OBJECTS {
      snmpSshtmSessionOpens,
      snmpSshtmSessionCloses,
      snmpSshtmSessionOpenErrors,
      snmpSshtmSessionUserAuthFailures,
      snmpSshtmSessionNoChannels,
      snmpSshtmSessionNoSubsystems,
      snmpSshtmSessionNoSessions,
      snmpSshtmSessionInvalidCaches
    }
    STATUS      current
    DESCRIPTION "A collection of objects for maintaining information
                 of an SNMP engine that implements the SNMP Secure
                 Shell Transport Model.
                "
```

```
    ::= { snmpSshtmGroups 2 }

END
```

8. Operational Considerations

The SSH Transport Model will likely not work in conditions where remote access to the CLI has stopped working. The SSH Transport Model assumes that TCP and IP continue to operate correctly between the communicating nodes. Failures in either node, death of the deamon serving the communication, routing problems in the network between, firewalls that block the traffic, and other problems can prevent the SSH Transport Model from working. In situations where management access has to be very reliable, operators should consider mitigating measures. These measures may include dedicated management-only networks, point-to-point links, and the ability to use alternate protocols and transports.

To have SNMP properly utilize the security services provided by SSH, the SSH Transport Model MUST be used with a Security Model that knows how to process a tmStateReference, such as the Transport Security Model for SNMP [RFC5591].

If the SSH Transport Model is configured to utilize AAA services, operators should consider configuring support for local authentication mechanisms, such as local passwords, so SNMP can continue operating during times of network stress.

The SSH protocol has its own window mechanism, defined in RFC 4254. The SSH specifications leave it open when window adjustment messages should be created, and some implementations send these whenever received data has been passed to the application. There are noticeable bandwidth and processing overheads to handling such window adjustment messages, which can be avoided by sending them less frequently.

The SSH protocol requires the execution of CPU-intensive calculations to establish a session key during session establishment. This means that short-lived sessions become computationally expensive compared to USM, which does not have a notion of a session key. Other transport security protocols such as TLS support a session-resumption feature that allows reusing a cached session key. Such a mechanism does not exist for SSH and thus SNMP applications should keep SSH sessions for longer time periods.

To initiate SSH connections, an entity must be configured with SSH client credentials plus information to authenticate the server. While hosts are often configured to be SSH clients, most

internetworking devices are not. To send notifications over SSHTM, the internetworking device will need to be configured as an SSH client. How this credential configuration is done is implementation- and deployment-specific.

9. Security Considerations

This memo describes a Transport Model that permits SNMP to utilize SSH security services. The security threats and how the SSH Transport Model mitigates those threats is covered in detail throughout this memo.

The SSH Transport Model relies on SSH mutual authentication, binding of keys, confidentiality, and integrity. Any authentication method that meets the requirements of the SSH architecture will provide the properties of mutual authentication and binding of keys.

SSHv2 provides perfect forward secrecy (PFS) for encryption keys. PFS is a major design goal of SSH, and any well-designed key-exchange algorithm will provide it.

The security implications of using SSH are covered in [RFC4251].

The SSH Transport Model has no way to verify that server authentication was performed, to learn the host's public key in advance, or to verify that the correct key is being used. The SSH Transport Model simply trusts that these are properly configured by the implementer and deployer.

SSH provides the "none" userauth method. The SSH Transport Model MUST NOT be used with an SSH connection with the "none" userauth method. While SSH does support turning off confidentiality and integrity, they MUST NOT be turned off when used with the SSH Transport Model.

The SSH protocol is not always clear on whether the user name field must be filled in, so for some implementations, such as those using GSSAPI authentication, it may be necessary to use a mapping algorithm to transform an SSH identity to a tmSecurityName or to transform a tmSecurityName to an SSH identity.

In other cases, the user name may not be verified by the server, so for these implementations, it may be necessary to obtain the user name from other credentials exchanged during the SSH exchange.

9.1. Skipping Public Key Verification

Most key-exchange algorithms are able to authenticate the SSH server's identity to the client. However, for the common case of Diffie-Hellman (DH) signed by public keys, this requires the client to know the host's public key a priori and to verify that the correct key is being used. If this step is skipped, then authentication of the SSH server to the SSH client is not done. Data confidentiality and data integrity protection to the server still exist, but these are of dubious value when an attacker can insert himself between the client and the real SSH server. Note that some userauth methods may defend against this situation, but many of the common ones (including password and keyboard-interactive) do not and, in fact, depend on the fact that the server's identity has been verified (so passwords are not disclosed to an attacker).

SSH MUST NOT be configured to skip public-key verification for use with the SSH Transport Model.

9.2. Notification Authorization Considerations

SNMP Notifications are authorized to be sent to a receiver based on the securityName used by the notification originator's SNMP engine. This authorization is performed before the message is actually sent and before the credentials of the remote receiver have been verified. Thus, the credentials presented by a notification receiver MUST match the expected value(s) for a given transport address, and ownership of the credentials MUST be properly cryptographically verified.

9.3. SSH User and Key Selection

If a "user@" prefix is used within an SnmpSSHAddress value to specify an SSH user name to use for authentication, then the key presented to the remote entity MUST be the key expected by the server for the "user". This may be different than a locally cached key identified by the securityName value.

9.4. Conceptual Differences between USM and SSHTM

The User-based Security Model [RFC3414] employed symmetric cryptography and user-naming conventions. SSH employs an asymmetric cryptography and naming model. Unlike USM, cryptographic keys will be different on both sides of the SSH connection. Both sides are responsible for verifying that the remote entity presents the right key. The optional "user@" prefix component of the SnmpSSHAddress Textual Convention allows the client SNMP stack to associate the connection with a securityName that may be different than the SSH user name presented to the SSH server.

9.5. The 'none' MAC Algorithm

SSH provides the "none" Message Authentication Code (MAC) algorithm, which would allow you to turn off data integrity while maintaining confidentiality. However, if you do this, then an attacker may be able to modify the data in flight, which means you effectively have no authentication.

SSH MUST NOT be configured using the "none" MAC algorithm for use with the SSH Transport Model.

9.6. Use with SNMPv1/v2c Messages

The SNMPv1 and SNMPv2c message processing described in [RFC3584] (BCP 74) always selects the SNMPv1 or SNMPv2c Security Models, respectively. Both of these and the User-based Security Model typically used with SNMPv3 derive the securityName and securityLevel from the SNMP message received, even when the message was received over a secure transport. Access control decisions are therefore made based on the contents of the SNMP message, rather than using the authenticated identity and securityLevel provided by the SSH Transport Model.

9.7. MIB Module Security

There are no management objects defined in this MIB module that have a MAX-ACCESS clause of read-write and/or read-create. So, if this MIB module is implemented correctly, then there is no risk that an intruder can alter or create any management objects of this MIB module via direct SNMP SET operations.

Some of the readable objects in this MIB module (i.e., objects with a MAX-ACCESS other than not-accessible) may be considered sensitive or vulnerable in some network environments. It is thus important to control even GET and/or NOTIFY access to these objects and possibly to even encrypt the values of these objects when sending them over the network via SNMP. These are the tables and objects and their sensitivity/vulnerability:

- The information in the snmpSshtmSession group is generated locally when a client session is being opened or closed. This information can reflect the configured capabilities of a remote SSH server, which could be helpful to an attacker for focusing an attack.

SNMP versions prior to SNMPv3 did not include adequate security. Even if the network itself is secure (for example by using IPSec or SSH), even then, there is no control as to who on the secure network is allowed to access and GET/SET (read/change/create/delete) the objects in this MIB module.

It is RECOMMENDED that implementers consider the security features as provided by the SNMPv3 framework (see [RFC3410], Section 8), including full support for cryptographic mechanisms for authentication and privacy, such as those found in the User-based Security Model [RFC3414], the Transport Security Model [RFC5591], and the SSH Transport Model described in this document.

Further, deployment of SNMP versions prior to SNMPv3 is NOT RECOMMENDED. Instead, it is RECOMMENDED to deploy SNMPv3 and to enable cryptographic security. It is then a customer/operator responsibility to ensure that the SNMP entity giving access to an instance of this MIB module is properly configured to give access to the objects only to those principals (users) that have legitimate rights to indeed GET or SET (change/create/delete) them.

10. IANA Considerations

IANA has assigned:

1. Two TCP port numbers in the Port Numbers registry that will be the default ports for the SNMP-over-SSH Transport Model as defined in this document, and the SNMP-over-SSH Transport Model for notifications as defined in this document. The assigned keywords and port numbers are "snmpssh" (5161) and "snmpssh-trap" (5162).

2. An SMI number (189) under mib-2, for the MIB module in this document.

3. An SMI number (7) under snmpDomains, for the snmpSSHDomain.

4. "ssh" as the corresponding prefix for the snmpSSHDomain in the SNMP Transport Domains registry; defined in [RFC5590].

5. "snmp" as a Connection Protocol Subsystem Name in the SSH Protocol Parameters registry.

11. Acknowledgments

The editors would like to thank Jeffrey Hutzelman for sharing his SSH insights, and Dave Shield for an outstanding job wordsmithing the existing document to improve organization and clarity.

Additionally, helpful document reviews were received from Juergen Schoenwaelder.

12. References

12.1. Normative References

[RFC1033] Lottor, M., "Domain administrators operations guide", RFC 1033, November 1987.

[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.

[RFC2579] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Textual Conventions for SMIv2", STD 58, RFC 2579, April 1999.

[RFC2580] McCloghrie, K., Perkins, D., and J. Schoenwaelder, "Conformance Statements for SMIv2", STD 58, RFC 2580, 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.

[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.

[RFC3418] Presuhn, R., "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3418, December 2002.

[RFC3490] Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications (IDNA)", RFC 3490, March 2003.

[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.

[RFC4252] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Authentication Protocol", RFC 4252, January 2006.

[RFC4253] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, January 2006.

[RFC4254] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Connection Protocol", RFC 4254, January 2006.

[RFC5590] Harrington, D. and J. Schoenwaelder, "Transport Subsystem for the Simple Network Management Protocol (SNMP)", RFC 5590, June 2009.

12.2. Informative References

[RFC1994] Simpson, W., "PPP Challenge Handshake Authentication Protocol (CHAP)", RFC 1994, August 1996.

[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.

[RFC3588] Calhoun, P., Loughney, J., Guttman, E., Zorn, G., and J. Arkko, "Diameter Base Protocol", RFC 3588, September 2003.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4256] Cusack, F. and M. Forssen, "Generic Message Exchange Authentication for the Secure Shell Protocol (SSH)", RFC 4256, January 2006.

[RFC4462] Hutzelman, J., Salowey, J., Galbraith, J., and V. Welch, "Generic Security Service Application Program Interface (GSS-API) Authentication and Key Exchange for the Secure Shell (SSH) Protocol", RFC 4462, May 2006.

[RFC4742] Wasserman, M. and T. Goddard, "Using the NETCONF Configuration Protocol over Secure SHell (SSH)", RFC 4742, December 2006.

[RFC5090] Sterman, B., Sadolevsky, D., Schwartz, D., Williams, D., and W. Beck, "RADIUS Extension for Digest Authentication", RFC 5090, February 2008.

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

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

Joseph Salowey
Cisco Systems
2901 3rd Ave
Seattle, WA 98121
USA

EMail: jsalowey@cisco.com

Wes Hardaker
Cobham Analytic Solutions
P.O. Box 382
Davis, CA 95617
US

Phone: +1 530 792 1913
EMail: ietf@hardakers.net