Secure Shell Transport Model for SNMP

draft-ietf-isms-secshell-07.txt

Status of This Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are draft documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on November 11, 2007.

Copyright Notice

Copyright (C) The IETF Trust (2007).

Abstract

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

Table of Contents

1. Introduction .................................................. 3
   1.1. The Internet-Standard Management Framework ............. 3
1. Introduction

This memo describes a Transport Model for the Simple Network Management Protocol, using the Secure Shell protocol within a transport subsystem [I-D.ietf-isms-tmsm]. 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 terms "manager" and "agent" are not used in this document, because in the RFC 3411 architecture [RFC3411], all SNMP entities have the capability of acting as either manager or agent or both depending on the SNMP applications included in the engine. 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.

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.

While SSH and USM frequently refer to a user, the terminology used in RFC3411 [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, with each acting in a particular role; an application or a set of applications, or a combination of these within an administrative domain.

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].

Sections requiring further editing are identified by [todo] markers in the text. Points requiring further WG research and discussion are identified by [discuss] markers in the text.

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" [I-D.ietf-isms-tmsm].

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 decisions to use self-contained documents, this memo includes the elements of procedure plus associated MIB objects which are needed for processing the Secure Shell Transport Model for SNMP. These MIB objects SHOULD NOT be referenced in other documents. This allows the Secure Shell Transport Model for SNMP to be designed and documented as independent and self-contained, having no direct impact on other modules, and this allows this module to be upgraded and supplemented as the need arises, and to move along the standards track on different time-lines from other modules.

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 typically utilizes a separate user and key management infrastructure. Operators have reported that deploying another user and key management infrastructure in order to use SNMPv3 is a reason for not deploying SNMPv3 at this point in time.

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 the time until deployment is possible.

The work will address the requirement for the SSH client to authenticate the SSH server, for the SSH server to authenticate the SSH client, and describe 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.

The work will include the ability to use any of the client authentication methods described in "SSH Authentication Protocol" [RFC4252]—public key, password, and host-based. Local accounts may be supported through the use of the public key, host-based or password based mechanisms. The password based mechanism allows for integration with deployed password infrastructure such as AAA servers using the RADIUS protocol [RFC2865]. The SSH Transport Model SHOULD be able to take advantage of other defined authentication mechanisms such as those defined in [RFC4462] and future mechanisms such as those that make use of X.509 certificate credentials. This will allow the SSH Transport Model to utilize client authentication and key exchange mechanisms which support different security infrastructures and provide different security properties.

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, using SSH user-auth [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 require no changes to the SNMP architecture.

5. A transport model should require no changes to the underlying 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 components:

- 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.
Internet-Draft    Secure Shell Transport Model for SNMP         May 2007

- 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.

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 plugs into the Transport Subsystem. 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. Message stream modification - SSH provides for verification that each received message has not been modified during its transmission through the network.
2. Information modification - SSH provides for verification that the contents of each received message has not been modified during its transmission through the network, data has not been altered or destroyed in an unauthorized manner, nor have data sequences been altered to an extent greater than can occur non-maliciously.

3. Masquerade - SSH provides for both verification of the identity of the SSH server and verification of the identity of the SSH client - the principal on whose behalf a received SNMP message claims to have been generated. It is not possible to assure the specific principal that originated a received SNMP message; rather, it is the principal on whose behalf the message was originated that is authenticated. SSH provides verification of the identity of the SSH server through the SSH Transport Protocol server authentication [RFC4253].

4. Verification of 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 will be used to map to an SNMP model-independent securityName for use with SNMP access control.

5. Authenticating both the SSH server and the SSH client ensures the authenticity of the SNMP engine that provides MIB data, whether that engine resides on the server or client side of the association. Operators or management applications might act upon the data they receive (e.g., raise an alarm for an operator, modify the configuration of the device that sent the notification, modify the configuration of other devices in the network as the result of the notification, and so on), so it is important to know that the provider of MIB data is authentic.

6. Disclosure - the SSH Transport Model provides that the contents of each received SNMP message are protected from disclosure to unauthorized persons.

7. Replay - SSH ensures that cryptographic keys established at the beginning of the SSH session and stored in the SSH session state are fresh new session keys generated for each session. These are used to authenticate and encrypt data, and to prevent replay across sessions. SSH uses sequence information to prevent the replay and reordering of messages within a session.

3.1.2. Data Origin Authentication Issues

The RFC 3411 architecture recognizes three levels of security:
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 the SSH Transport Model provides this information to SSH for an outgoing message. The 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.

The SSH Transport Model does not know about the algorithms or options to open SSH sessions that match different securityLevels. For interoperability of the trust assumptions between SNMP engines, 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 requested by SNMP applications and specified with a lesser securityLevel (noAuthNoPriv or authNoPriv) are sent by the SSH Transport Model as authPriv securityLevel.

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

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 - and 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 CHAP [RFC1994] or digest authentication [RFC 2617, draft-ietf-radext-digest-auth-04] to integrate with RADIUS or Diameter. These mechanisms leave the password in the clear on the device that is authenticating the password which introduces threats to the authentication infrastructure.

GSSKeyex [RFC4462] provides a framework for the addition of client authentication mechanisms which 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. Privacy Protocol

The SSH transport layer protocol provides strong encryption, server authentication, and integrity protection.

3.1.5. Protection against Message Replay, Delay and Redirection

SSH uses sequence numbers and integrity checks to protect against replay and reordering of messages within a connection.

SSH also provides protection against replay of entire sessions. In a properly-implemented Diffie-Helman exchange, both sides will generate new random numbers for each exchange, which means the exchange hash and thus the encryption and integrity keys will be distinct for every session.

3.1.6. SSH Subsystem

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

SSH subsystems of type "snmp" are 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 or 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 transportAddressType, transportAddress, securityName, and securityLevel associated with each session.

### 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, and 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.

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 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 should be sent is typically determined and configured by a network administrator. The SNMP-TARGET-MIB module [RFC3413] contains objects for defining management targets, including transport domains and addresses and security parameters, for applications such as notifications and proxy.

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 perform proxy.
These 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. Passing Security Parameters

For the SSH Transport Model, the session state needs to be maintained
using tmStateReference. RFC3411 discusses a securityStateReference,
but this is not accessible to the Transport Subsystem.

4.1. tmStateReference

Upon opening each SSH connection, the SSH Transport Model stores
model- and mechanism-specific information about the connection in a
cache, referenced by tmStateReference.

An implementation might store the contents of the cache in a Local
Configuration Datastore (LCD). For ease of understanding, this
document represents the SSH Transport Model specific cache and LCD as
an SNMP-SSH-TM-MIB module.

```
tmTransport = snmpSSHDomain

tmAddress = a snmpSSHAddress

tmSecurityLevel = "authPriv"

tmSecurityName = the principal name authenticated by SSH. How
this data is extracted from the SSH environment and how it is
translated into a securityName is implementation-dependent. By
default, the tmSecurityName is the name that has been successfully
authenticated by SSH, from the user name field of the
SSH_MSG_USERAUTH_REQUEST message.

tmSameSession = for outgoing messages, to indicate whether the
same session must be used for the outgoing message as was used for
the corresponding incoming message.

sshtmSessionID = [DISCUSS: SSH transport session identifier]
```

How the SSH identity is extracted from the SSH layer, and how the SSH
identity is mapped to a securityName is implementation-dependent.
Additional information may be stored in a local datastore (such as a
preconfigured mapping table) or in a cache, such as the value of an
SSH session identifier (as distinct from an SNMP session).

The tmStateReference is used to pass references containing the
appropriate SSH session information from the transport model for subsequent processing.

The SSH Transport Model has the responsibility for explicitly releasing the complete tmStateReference and deleting the associated information when the session is destroyed.

5. Elements of Procedure

Abstract service interfaces have been defined by RFC 3411 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. These RFC 3411-defined data flows are referred to here as public interfaces.

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 may return an OID and value for an incremented counter and a value for securityLevel, and values for contextEngineID and contextName for the counter, and the securityStateReference if the information is available at the point where the error is detected.

[DISCUSS: does transport model ever have access to contextEngineID?]

5.1. Procedures for an Incoming Message

For an incoming message, the SSH Transport Model will put information from the SSH layer into a cache referenced by tmStateReference.

1) The SSH Transport Model queries the associated SSH engine, in an implementation-dependent manner, to determine the transport and security parameters for the received message.

   tmTransportDomain = snmpSSHDomain
   tmTransportAddress = a snmpSSHAddress
   tmSecurityLevel = "authPriv"
tmsSecurityName = the principal name authenticated by SSH. How this data is extracted from the SSH environment and how it is translated into a securityName is implementation-dependent. By default, the tmSecurityName is the name that has been successfully authenticated by SSH, from the user name field of the SSH_MSG_USERAUTH_REQUEST message.

2) If one does not exist, the SSH Transport Model creates an entry in a Local Configuration Datastore, in an implementation-dependent format, containing the information and any implementation-specific parameters desired, and creates a tmStateReference for subsequent reference to the information.

Then the Transport model passes the message to the Dispatcher using the following primitive:

```plaintext
statusInformation =
receiveMessage(
  IN transportDomain       -- domain for the received message
  IN transportAddress      -- address for the received 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 an Outgoing Message

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

```plaintext
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) Determine the target index by extracting the transportDomain, transportAddress, securityName, and securityLevel from the tmStateReference.

2) Lookup the session in the Local Configuration Datastore using the target index
3) If there is no session open associated with the target index, then call openSession().

3a) If an error is returned from OpenSession(), then discard the message and return the error indication in the statusInformation.

3b) If openSession() is successful, then store any implementation-specific information in the LCD for subsequent use.

4) Extract any implementation-specific parameters from the LCD

5) Pass the wholeMessage to SSH for encapsulation in an SSH_MSG_CHANNEL_DATA message.

5.3. Establishing a Session

The Secure Shell Transport Model provides the following primitive to describe the data passed between the Transport Model and the SSH service. It is an implementation decision how such data is passed.

statusInformation =
openSession(
IN  destTransportDomain -- transport domain to be used
IN  destTransportAddress -- transport address to be used
IN  securityName -- on behalf of this principal
IN  securityLevel -- Level of Security requested
IN  maxMessageSize -- of the sending SNMP entity
OUT tmStateReference -- (NEW) transport info
)

The following describes the procedure to follow to establish a session between a client and server to run SNMP over SSH. This process is followed 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 or a Notification Originator or a Proxy Forwarder.

1) Using destTransportDomain and destTransportAddress, 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 parameters of the transport connection and the credentials used to authenticate are provided in an implementation-dependent manner.

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

2) The provided transport domain, transport address, securityName and securityLevel are used to lookup an associated entry in the Local Configuration Datastore (LCD). Any model-specific information concerning the principal at the destination is extracted. This step allows preconfiguration of model-specific principals mapped to the transport/name/level, for example, for sending notifications. Set the username in the SSH_MSG_USERAUTH_REQUEST to the username extracted from the LCD.

If information about the principal is absent from the LCD, then set the username in the SSH_MSG_USERAUTH_REQUEST to the value of securityName. This allows a deployment without preconfigured mappings between model-specific and model-independent names, but the securityName will need to contain a username recognized by the authentication mechanism.

3) The client will then invoke the "ssh-userauth" service to authenticate the user, as described in the SSH authentication protocol [RFC4252]. The credentials used to authenticate are provided in an implementation-dependent manner.

If the authentication is unsuccessful, then the transport connection is closed, tmStateReference is released, the message is discarded, an error indication (unknownSecurityName) is returned to the calling module, and processing stops for this message.

4) Once the principal has been successfully authenticated, the client will invoke the "ssh-connection" service, also known as the SSH connection protocol [RFC4254].

5) After the ssh-connection service is established, the client will use an SSH_MSG_CHANNEL_OPEN message to open a channel of type "session", providing a selected sender channel number, and a maximum packet size calculated from the SNMP maxMessageSize.

6) If successful, this will result in an SSH session. The destTransportDomain and the destTransportAddress, plus the "recipient channel" and "sender channel" and other relevant data from the SSH_MSG_CHANNEL_OPEN_CONFIRMATION should be retained so they can be added to the LCD for subsequent use.

7) Once the SSH session has been established, the client will invoke SNMP as an SSH subsystem, as indicated in the "subsystem" parameter.
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 port. Servers SHOULD be configurable to allow access to the SNMP SSH subsystem over other ports.

8) Create an entry in a Local Configuration Datastore containing the provided transportDomain, transportAddress, securityName, securityLevel, and SSH-specific parameters and create a tmStateReference to reference the entry.

5.4. Closing a Session

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

statusInformation = closeSession(
  IN  tmStateReference        -- transport info
)

The following describes the procedure to follow to close a session between a client and server. This process is followed by any SNMP engine closing the corresponding SNMP session.

1) Extract the transportDomain, transportAddress, securityName, and securityLevel from the tmStateReference.

2) Lookup the session in the Local Configuration Datastore using the target index

3) If there is no session open associated with the target index, then closeSession processing is completed.

4) Extract any implementation-specific parameters from the LCD

5) Have SSH close the specified session.

6. MIB Module Overview

This MIB module provides management of the Secure Shell Transport Model. It defines some needed textual conventions, and some statistics.
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. The sshtmStats Subtree

This subtree contains SSH transport-model-dependent counters.

This subtree provides information for identifying fault conditions and performance degradation.

6.4. The sshtmUserTable

This table contains SSH Transport Model information about SSH principals.

6.5. 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], the SNMP-FRAMEWORK-MIB [RFC3411] and the SNMP-TRANSPORT-MIB [I-D.ietf-isms-tmsm].

This MIB module is for managing SSH Transport Model information. This MIB module models a sample Local Configuration Datastore.

6.5.1. MIB Modules Required for IMPORTS

The following MIB module imports items from [RFC2578], [RFC2579], [RFC2580], [RFC3411], [RFC3419], and [I-D.ietf-isms-tmsm]

This MIB module also references [RFC3490]

7. MIB module definition

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

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE,
OBJECT-IDENTITY, snmpModules, snmpDomains
    FROM SNMPv2-SMI
TestAndIncr, TEXTUAL-CONVENTION,
StorageType, RowStatus
    FROM SNMPv2-TC
MODULE-COMPLIANCE, OBJECT-GROUP
    FROM SNMPv2-CONF
SnmpAdminString
    FROM SNMP-FRAMEWORK-MIB
TransportAddress, TransportAddressType
    FROM TRANSPORT-ADDRESS-MIB
;

sshtmMIB MODULE-IDENTITY
    LAST-UPDATED "200610050000Z"
    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
    International University Bremen
    Campus Ring 1
    28725 Bremen
    Germany
    +49 421 200-3587
    j.schoenwaelder@iu-bremen.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
DESCRIPTION "The Secure Shell Transport Model MIB

Copyright (C) The Internet Society (2006). This version of this MIB module is part of RFC XXXX; see the RFC itself for full legal notices.

-- NOTE to RFC editor: replace XXXX with actual RFC number
-- for this document and remove this note

REVISION "200610050000Z" -- 02 September 2005
DESCRIPTION "The initial version, published in RFC XXXX
-- NOTE to RFC editor: replace XXXX with actual RFC number
-- for this document and remove this note

 ::= { snmpModules xxxx }
-- RFC Ed.: replace xxxx with IANA-assigned number and
-- remove this note

-- ---------------------------------------------------------------------
-- sub-trees in the SNMP-SSH-TM-MIB
-- ---------------------------------------------------------------------

sshtmNotifications OBJECT IDENTIFIER ::= { sshtmMIB 0 }
sshtmMIBObjects    OBJECT IDENTIFIER ::= { sshtmMIB 1 }
sshtmConformance   OBJECT IDENTIFIER ::= { sshtmMIB 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."
 ::= { snmpDomains yy }

Harrington & Salowey Expires November 11, 2007 [Page 20]
SnmpSSHAddress ::= TEXTUAL-CONVENTION
   DISPLAY-HINT "1a"
   STATUS      current
   DESCRIPTION
           "Represents either a hostname encoded in ASCII
           using the IDNA protocol, as specified in RFC3490, followed by
           a colon ‘:’ (ASCII character 0x3A) and a decimal port number
           in ASCII, or an IP address followed by a colon ‘:’
           (ASCII character 0x3A) and a decimal port number in ASCII.
           The name SHOULD be fully qualified whenever possible.

           Values of this textual convention are not directly useable
           as transport-layer addressing information, and 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 specified in SMIV2, STD 58. In this case,
           the OBJECT-TYPE declaration MUST include a ‘SIZE’ clause
           to limit the number of potential instance sub-identifiers."
   SYNTAX      OCTET STRING (SIZE (1..255))

-- The sshtmSession Group

sshtmSession OBJECT IDENTIFIER ::= { sshtmMIBObjects 1 }

sshtmSessionCurrent OBJECT-TYPE
   SYNTAX       Gauge32
MAX-ACCESS   read-only
STATUS       current
DESCRIPTION "The current number of open sessions.
"
::= { sshtmSession 1 }

sshtmSessionMaxSupported OBJECT-TYPE
SYNTAX       Unsigned32
MAX-ACCESS   read-only
STATUS       current
DESCRIPTION "The maximum number of open sessions supported. The value zero indicates the maximum is dynamic.
"
::= { sshtmSession 2 }

sshtmSessionOpenErrors OBJECT-TYPE
SYNTAX       Counter32
MAX-ACCESS   read-only
STATUS       current
DESCRIPTION "The number of times an openSession() request failed to open a Session.
"
::= { sshtmSession 3 }

sshtmSessionNoAvailableSessions OBJECT-TYPE
SYNTAX       Counter32
MAX-ACCESS   read-only
STATUS       current
DESCRIPTION "The number of times a Response message was dropped because the corresponding session was no longer available.
"
::= { sshtmSession 5 }

-- The sshtmUser Group ********************************************

sshtmUser OBJECT IDENTIFIER ::= { sshtmMIBObjects 2 }

sshtmUserSpinLock OBJECT-TYPE
SYNTAX       TestAndIncr
MAX-ACCESS   read-write
STATUS       current
DESCRIPTION "An advisory lock used to allow several cooperating Command Generator Applications to coordinate their use of facilities to alter the sshtmUserTable.
"
::= { sshtmUser 1 }
sshtmUserTable OBJECT-TYPE
SYNTAX SEQUENCE OF SshtmUserEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION "The table of users configured in the SNMP engine’s Local Configuration Datastore (LCD).

Most configuration of this table is expected to be done by an agent dynamically. It is possible for an SNMP management application to pre-configure the table with static information useful for translating from an SSH-specific user to a model-independent securityName, or for statically configuring the only entities authorized to receive notifications.

To create a new user (i.e., to instantiate a new conceptual row in this table), it is recommended to follow this procedure:

1) GET(sshtmUserSpinLock.0) and save in sValue.
2) SET(sshtmUserSpinLock.0=sValue,
   sshtmUserStatus=createAndWait)
3) configure the entry
4) SET(sshtmUserStatus=active)

The new user should now be available and ready to be used for SNMPv3 communication.

The use of sshtmUserSpinlock is to avoid conflicts with another SNMP command generator application which may also be acting on the sshtmUserTable.

::= { sshtmUser 2 }

sshtmUserEntry OBJECT-TYPE
SYNTAX SshtmUserEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION "A user configured in the SNMP engine’s Local Configuration Datastore (LCD) for the SSH Transport Model.

INDEX { sshtmUserAddress,
   sshtmUserName
   }
::= { sshtmUserTable 1 }
SshtmUserEntry ::= SEQUENCE
  {
    sshtmUserAddress          snmpSSHAddress,
    sshtmUserSecurityName     SnmpAdminString,
    sshtmUserName             SnmpAdminString,
    sshtmUserStorageType      StorageType,
    sshtmUserStatus           RowStatus
  }

sshtmUserAddress  OBJECT-TYPE
  SYNTAX       snmpSSHAddress
  MAX-ACCESS   not-accessible
  STATUS       current
  DESCRIPTION "A remote SNMP engine’s SSH address.
          
          ::= { sshtmUserEntry 1 }

sshtmUserSecurityName OBJECT-TYPE
  SYNTAX       SnmpAdminString
  MAX-ACCESS   read-only
  STATUS       current
  DESCRIPTION "A human readable string representing the user in
            Transport Model independent format.

            The default transformation of the sshtmUserName to
            the sshtmUserSecurityName and vice versa is the
            identity function so that the sshtmUserSecurityName
            is usually the same as the sshtmUserName.
          
          ::= { sshtmUserEntry 2 }

sshtmUserName      OBJECT-TYPE
  SYNTAX       SnmpAdminString (SIZE(1..32))
  MAX-ACCESS   not-accessible
  STATUS       current
  DESCRIPTION "This is the user name used in the
            SSH_MSG_USERAUTH_REQUEST to authenticate the client.
          
          ::= { sshtmUserEntry 3 }

sshtmUserStorageType OBJECT-TYPE
  SYNTAX       StorageType
  MAX-ACCESS   read-create
  STATUS       current
  DESCRIPTION "The storage type for this conceptual row.

          It is an implementation issue to decide if a SET for
a readOnly or permanent row is accepted at all. In some contexts this may make sense, in others it may not. If a SET for a readOnly or permanent row is not accepted at all, then a ‘wrongValue’ error must be returned.

DEFVAL { nonVolatile }
 ::= { sshtmUserEntry 4 }

sshtmUserStatus OBJECT-TYPE
SYNTAX RowStatus
MAX-ACCESS read-create
STATUS current
DESCRIPTION "The status of this conceptual row. Until instances of all corresponding columns are appropriately configured, the value of the corresponding instance of the sshtmUserStatus column is ‘notReady’.

The value of this object has no effect on whether other objects in this conceptual row can be modified." ::= { sshtmUserEntry 5 }

-- ************************************************
-- sshtmMIB - Conformance Information
-- ************************************************

sshtmGroups OBJECT IDENTIFIER ::= { sshtmConformance 1 }

sshtmCompliances OBJECT IDENTIFIER ::= { sshtmConformance 2 }

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

sshtmGroup OBJECT-GROUP
OBJECTS {
  sshtmUserSpinLock,
  sshtmUserSecurityName,
  sshtmUserStorageType,
  sshtmUserStatus
}
STATUS current
DESCRIPTION "A collection of objects for maintaining information of an SNMP engine which implements the SNMP Secure Shell Transport Model."
 organizational considerations

The SSH Transport Model will likely not work in conditions where access to the CLI has stopped working. In situations where SNMP access has to work when the CLI has stopped working, a UDP transport model should be considered instead of the SSH Transport Model.

The SSH Transport Model defines a single well-known default port for all traffic types. Administrators might choose to define one port for SNMP request-response traffic, but configure notifications to be sent to a different port, using the snmpTargetAddrTable.

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

The SSH protocol has its own windowing mechanism. RFC 4254 says: The window size specifies how many bytes the other party can send before it must wait for the window to be adjusted. Both parties use the following message to adjust the window. The SSH specifications leave it open when such window adjustment messages are created. Some implementations have been found to send window adjustment messages whenever received data has been passed to the application. Since window adjustment messages are padded, encrypted, hmac’ed, and wrapped, this results in noticeable bandwidth and processing overhead, which can be avoided by sending window adjustment messages 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.

9. Security Considerations

This document 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. While SSH does support turning off confidentiality and integrity, they SHOULD NOT be turned off when used with the SSH Transport Model.

SSHv2 provides Perfect Forward Security (PFS) for encryption keys. PFS is a major design goal of SSH, and any well-designed keyex 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 verify that the correct key is being used. the SSH Transport Model simply trusts that these are properly configured by the implementer and deployer.

9.1. noAuthPriv

SSH provides the "none" userauth method, which is normally rejected by servers and used only to find out what userauth methods are supported. However, it is legal for a server to accept this method, which has the effect of not authenticating the SSH client to the SSH server. Doing this does not compromise authentication of the SSH server to the SSH client, nor does it compromise data confidentiality or data integrity.

SSH supports anonymous access. If the SSH Transport Model can extract from SSH an authenticated principal to map to securityName, then anonymous access SHOULD be supported. It is possible for SSH to skip entity authentication of the client through the "none"
authentication method to support anonymous clients, however in this case an implementation MUST still support data integrity within the SSH transport protocol and provide an authenticated principal for mapping to securityName for access control purposes.

The RFC 3411 architecture does not permit noAuthPriv. The SSH Transport Model SHOULD NOT be used with an SSH connection with the "none" userauth method.

9.2. Use with SNMPv1/v2c messages

The SNMPv1 and SNMPv2c message processing described in RFC3584 (BCP 74) always selects the SNMPv1(1) Security Model for an SNMPv1 message, or the SNMPv2c(2) Security Model for an SNMPv2c message. When running SNMPv1/SNMPv2c over a secure transport like the SSH Transport Model, the securityName and securityLevel used for access control decisions are then derived from the community string, not the authenticated identity and securityLevel provided by the SSH Transport Model.

9.3. 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 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.

9.4. the ‘none’ MAC algorithm

SSH provides the "none" 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.
9.5. MIB module security

There are a number of management objects defined in this MIB module with a MAX-ACCESS clause of read-write and/or read-create. Such objects may be considered sensitive or vulnerable in some network environments. The support for SET operations in a non-secure environment without proper protection can have a negative effect on network operations. These are the tables and objects and their sensitivity/vulnerability:

- [todo]

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:

- [todo]

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 the USM and the SSH Transport Model cryptographic mechanisms (for authentication and privacy).

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 is requested to assign:

1. a TCP port number in the range 1..1023 in the http://www.iana.org/assignments/port-numbers registry which will be the default port for SNMP over an SSH Transport Model as defined in this document,

2. an SMI number under snmpModules, for the MIB module in this document,

3. an SMI number under snmpDomains, for the snmpSSHDomain,

4. "snmp" as an SSH Service Name in the http://www.iana.org/assignments/ssh-parameters registry.

11. Acknowledgements

The editors would like to thank Jeffrey Hutzelman for sharing his SSH insights.

12. References

12.1. Normative References


Internet-Draft    Secure Shell Transport Model for SNMP         May 2007


12.2. Informative References


Appendix A. Open Issues

We need to reach consensus on some issues.

Here is the current list of issues from the SSH Transport Model document where we need to reach consensus.

- The MIB module needs to be finalized.
- Consistency with TSM needs to be done
- SSH transport domain and transport address definitions - consistency across WGs
o configuring notification originators

TODO:

finalize error processing in EOP

develop sameSession support

update MIB security considerations

Appendix B. Change Log

From -06- to -07

removed section on SSH to EngineID mappings, since engineIDs are not exposed to the transport model

removed references to engineIDs and discovery

removed references to securityModel.

added security considerations warning about using with SNMPv1/v2c messages.

added keyboard interactive discussion

noted some implementation-dependent points

removed references to transportModel; we use the transport domain as a model identifier.

cleaned up ASIs

modified MIB to be under snmpModules

changed transportAddressSSH to snmpSSHDomain style addressing

From -05- to -06

replaced transportDomainSSH with RFC3417-style snmpSSHDomain

replaced transportAddressSSH with RFC3417-style snmpSSHAddress

Changed recvMessage to receiveMessage, and modified OUT to IN to match TMSM.

From -04- to -05
added sshtmUserTable

moved session table into the transport model MIB from the transport subsystem MIB

added and then removed Appendix A - Notification Tables
Configuration (see Transport Security Model)

made this document a specification of a transport model, rather than a security model in two parts. Eliminated TMSP and MPSP and replaced them with "transport model" and "security model".

Removed security-model-specific processing from this document.

Removed discussion of snmpv3/v1/v2c message format co-existence

changed tmSessionReference back to tmStateReference

"From -03- to -04-"

changed tmStateReference to tmSessionReference

"From -02- to -03-"

rewrote almost all sections

merged ASI section and Elements of Procedure sections

removed references to the SSH user, in preference to SSH client

updated references

created a conventions section to identify common terminology.

rewrote sections on how SSH addresses threats

rewrote mapping SSH to engineID

eliminated discovery section

detailed the Elements of Procedure

eliminated sections on msgFlags, transport parameters

resolved issues of opening notifications
eliminated sessionID (TMSM needs to be updated to match)

eliminated use of tmsSessionTable except as an example

updated Security Considerations

"From -01- to -02-"

Added TransportDomainSSH and Address

Removed implementation considerations

Changed all "user auth" to "client auth"

Removed unnecessary MIB module objects

updated references

improved consistency of references to TMSM as architectural extension

updated conventions

updated threats to be more consistent with RFC3552

discussion of specific SSH mechanism configurations moved to security considerations

modified session discussions to reference TMSM sessions

expanded discussion of engineIDs

wrote text to clarify the roles of MPSP and TMSP

clarified how snmpv3 message parts are ised by SSHSM

modified nesting of subsections as needed

securityLevel used by the SSH Transport Model always equals authpriv

removed discussion of using SSHSM with SNMPv1/v2c

started updating Elements of Procedure, but realized missing info needs discussion.

updated MIB module relationship to other MIB modules
"From -00- to -01-"

-00- initial draft as ISMS work product:

updated references to SecSH RFCs

Modified text related to issues# 1, 2, 8, 11, 13, 14, 16, 18, 19, 20, 29, 30, and 32.

updated security considerations

removed Juergen Schoenwaelder from authors, at his request

ran the mib module through smilint

Authors’ Addresses

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

Phone: +1 603 436 8634
EMail: dharrington@huawei.com

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

EMail: jsalowey@cisco.com