Network Working Group
Internet-Draft
Expires: February 26, 2004

E. Lear
Cisco Systems
T. Goddard
Wind River
S. Waldbusser
August 28, 2003

The SSH Protocol Mapping for NETCONF
draft-lear-netconf-over-ssh-00

Status of this Memo

Copyright Notice

Abstract

This document specifies a protocol mapping for NETCONF over Secure Shell (SSH). It is incomplete and informal in places as of yet, but should hopefully discuss raise and discuss the key issues that a transport mapping must address.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [1].

Table of Contents

1. Introduction

The NETCONF protocol [2] defines a mechanism to manage the configuration of network elements. NETCONF is designed to be usable over a variety of protocols. This document specifies a mapping over Secure Shell (ssh). [5][3]

Many operators have requested a transport mapping over SSH. This mapping addresses the needs of those who want a programmatic interface to the device. As such, we take advantage of SSH's ability to call a process other than that of the command line interface (CLI). This avoids interaction with prompts and banners that traditionally cause one's scripts to blow up. This is not to say that one could not implement an interface to call the netconf function from the CLI for debugging purposes, but that function need not be standardized.

2. The SSH Protocol Mapping

[5] provides for a secure transport where the listener is verified through use of a host key. A key point to note about SSH is that server authentication is required. NETCONF listeners must therefore be configured with a server key, and NETCONF initiators must have some intelligent means for determining whether to trust the listener's key.

A conforming manager must implement initiator functionality and a conforming agent must implement listener functionality. It is also mandatory for a manager to implement listener functionality and for an agent to implement initiator functionality.

[4] provides for user authentication and specifies several ways for the initiator to authenticate to the listener.

There are two possible ways to map Netconf to SSH. The first is to use the channel mechanism with SSH, thus only using one transport connection. The second method involves use of multiple transport connections where session information is shared between them. In both cases no shell access is used.

2.1 Multiple channels per transport

[6] provides for a channel-based connection protocol that runs atop the SSH transport and authentication protocols. The initiator will send SSH_MSG_CHANNEL_REQUEST with the "subsystem" string as described in Section 4.5 of that document. The subsystem name string MUST be "netconf-control" to initiate the netconf protocol. This channel maps to Netconf's control channel. A second channel request is then made with string name "netconf-data" to initiate the Netconf Data channel. The process of channel creation from this point forth follows as channels are necessary above NETCONF.

A potential problem with this approach is that scripting environments typically interface with SSH clients through stdio and will likely only be able to access the control channel through this interface. Thus scripting will be difficult until special-purpose tools are developed.

2.2 Multiple TCP transports

Just as with multiple channels, a transport connection is opened and authentication occurs. However, the mapping for <rpc> requests is somewhat more involved. In addition, since multiple connections are being used, it is necessary to indicate which set of connections is part of the same netconf session. Thus, a greeting is exchanged that

will indicate whether this is a new netconf session or if it is part of an existing netconf session.

Unlike multiple sessions per stream, with each case here, a single channel is opened with the name "netconf". Once the channel is established, a greeting is sent from the initiator to the listener, containing a 64 bit number "SessionID". If the value of the SessionId is 0, the listener will return a random 64 bit number that is not currently being used by another SSH session. If the value is greater than 0, then the listener will return that value or an error indicating that the session is unknown:

```
Initiator:
    <connect>
      <sessionid>34567</sessionid>
    <connect>

Listener:
    <connect>
      <sessionid>34567</sessionid>
    <connect>

  or:

    <connect>
      <error>Session not found</error>
    <connect>
```

The Session ID (SID) is used by the listener to group SSH transport connections into Netconf sessions. When a netconf channel is required, a new transport connection is established. The 5-tuple of the new transport connection is communicated over the initial channel by the listener to the initiator to indicate that the channel is ready for use.

When any SSH transport connection is brought up, each side must begin by sending a <hello> message advertising it's capabilities. Subsequently, any SSH transport connection can accept messages intended for the management channel, operations channel, or notification channel.

All transport connections are opened by the initiator, ensuring that the same level of access through firewalls is used for each additional transport request.

There are some obvious implementation challenges with this approach. The largest problem is that a communications channel may have to be handed off to different processes on the listener. Because SSH

communications terminate on a well known port, typically a single process dispatches all requests to that port. The other option of sending port information has substantial firewall implications. Picking random ports to listen on has shown itself to be difficult in practice. The other option of reversing the connection so that the listener initiates transport connections raises similar firewall concerns. In these last two cases, levels of access through a firewall differ between the initial connection and subsequent ones.

Another potential problem with this approach is that because reauthentication is required for every channel, user action may similarly be required, and authentication of a portion of a Netconf session may fail.

Another possible approach to managing multiple connections would be to use a separate well known port in which communications are encrypted with a session key that is established with the first SSH connection. If this is done, however, the utility of SSH is significantly reduced.

2.3 Directionality of Connections

Netconf presumes reversability. In order to provide for reversability in SSH, both the agent and the manager must have well known host keys. The listener will authenticate itself to the initiator with its host key. The initiator will then authenticate itself with any allowable mechanism, as specified by [4]. The authenticated principle is then passed to Netconf.

At this point netconf capabilities are exchanged on the control channel, and the protocol proceeds.

3. Operation Pipelining

A sequence of operations may be sent on a single connection (without waiting for replies to earlier operations), and these operations will be processed sequentially by the agent. It is recommended that message-ids be used to assist the management application in associating operations with replies. Each distinct operation must begin on a new line.

4. NETCONF Sessions

As a NETCONF session may be composed of either multiple TCP transport connections or multiple SSH channels, it is important to consider what state is maintained in sessions and how sessions are torn down.

4.1 Session State

A number NETCONF session state elements are common across the SSH channels or TCP transports associated with the session.

1. Authentication Information

2. Session ID (SID)

3. Capability Information

4. Pending Operations

5. Operation message-ids

A given NETCONF session has a single authenticated user mapped to an identity in the local database. That user may have multiple concurrent NETCONF sessions, but the given session has a single 64 bit Session ID (unique to the NETCONF agent). Capabilities must remain constant for the duration of a session to ensure consistent behaviour of management software, and are therefore common across any channels or transports in the session. At any point in time, a NETCONF session may have operations that have not yet completed. Such operations may have opaque message-ids, which have session scope and can be used on one connection to request <rpc-abort> for an operation initiated on another connection.

4.2 Session Teardown

Clean session teardown is an important feature of NETCONF and must be supported in the single and multiple connection SSH mappings. A session may be torn down in two ways: via the <kill-session> operation or via connection closure. (The <kill-session> operation will tear down a session and forcibly close any associated connections.)

If there is a single connection (SSH channel or TCP transport connection), the associated session is torn down when that connection is closed. If there are multiple connections associated with a session, then the session is torn down when every connection in that session is closed.

5. (Additional) Security Considerations

Configuration information is sensitive by its nature. The use of SSH provides for a secure means to provide authentication and encryption of communications. The keys used by devices and users should always be protected.

Because each connection is authenticated, either method mentioned in this document should provide about the same amount of safety.

6. Acknowledgments

The authors of this draft belatedly became aware of similar work done by Margaret Wasserman. There-in she takes a slightly different approach by limiting Netconf functionality in order to allow for a single stream.

This document was written with xml2rfc version 1.20.

Normative References

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

[2] Enns, R., "NETCONF Configuration Protocol", draft-ietf-netconf-prot-00 (work in progress), August 2003.

[3] Ylonen, T., Kivinen, T., Saarinen, M., Rinne, T. and S. Lehtinen, "SSH Protocol Architecture", draft-ietf-secsh-architecture-14 (work in progress), July 2003.

[4] Ylonen, T., Kivinen, T., Saarinen, M., Rinne, T. and S. Lehtinen, "SSH Authentication Protocol", draft-ietf-secsh-userauth-17 (work in progress), July 2003.

[5] Ylonen, T., Kivinen, T., Saarinen, M., Rinne, T. and S. Lehtinen, "SSH Transport Layer Protocol", draft-ietf-secsh-transport-16 (work in progress), July 2003.

[6] Ylonen, T., Kivinen, T., Rinne, T. and S. Lehtinen, "SSH Connection Protocol", draft-ietf-secsh-connect-17 (work in progress), July 2003.

Authors' Addresses

Eliot Lear
Cisco Systems
170 W. Tasman Dr.
San Jose, CA 95134
US

Phone: +1 408 527 4020
EMail: lear@cisco.com

Ted Goddard
Wind River
#180, 6815-8th St. N.E.
Calgary, AB T2E 7H7
Canada

Phone: +1 403 730 7590
EMail: ted.goddard@windriver.com

Steve Waldbusser

Phone: +1 650 948 6500
EMail: waldbusser@nextbeacon.com

Intellectual Property Statement

Full Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.