NETCONF Working Group
Internet-Draft
Updates: 4253 (if approved)
Intended status: Standards Track
Expires: December 20, 2013

K. Watsen
Juniper Networks

June 18, 2013

Reverse Secure Shell (Reverse SSH)
draft-ietf-netconf-reverse-ssh-00

Abstract

This memo presents a technique for a NETCONF server to initiate a SSH connection to a NETCONF client. This is accomplished by the NETCONF client listening on IANA-assigned TCP port XXX and starting the SSH client protocol immediately after accepting a TCP connection on it. This role-reversal is necessary as the NETCONF server must also be the SSH Server, in order for the NETCONF client to open the IANA-assigned SSH subsystem "netconf".

Status of This Memo

Copyright Notice

Table of Contents

1. Requirements Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Introduction

This memo presents a technique for a NETCONF [RFC6241] server to initiate a Secure Shell (SSH) [RFC4251] connection to a NETCONF client. This is accomplished by the NETCONF client listening on IANA-assigned TCP port XXX and starting the SSH client protocol immediately after accepting a TCP connection on it. This role-reversal is necessary as the NETCONF server must also be the SSH Server, in order for the NETCONF client to open the IANA-assigned SSH subsystem "netconf" [RFC6242].

While the motivation for this work is for the NETCONF protocol, the solution is not specific to NETCONF and is applicable any time it is desired for a SSH server to initiate a connection to a SSH client. For this reason, the solution is given the generic name "Reverse SSH" and the port the remote peer listens on is the Reverse SSH port.

3. Benefits to Device Management

The SSH protocol is nearly ubiquitous for device management, as it is the transport for the command-line applications `ssh`, `scp`, and `sftp` and is the required transport for the NETCONF protocol [RFC6241]. However, all these SSH-based protocols expect the managed device to be the SSH server.

Reverse SSH enables the managed device to consistently be the SSH server regardless of which peer initiates the underlying TCP connection. Maintaining the role of SSH Server is both necessary and desirable. It is necessary because SSH channels and subsystems can only be opened on the SSH Server. It is desirable because it conviently leverages infrastructure that may be deployed for host-key verification and user authentication.

Reverse SSH is useful for both initial deployment and on-going device management and may be used to enable any of the following scenarios:

- o The device may proactively "call home" after being powered on for the first time to register itself with its management system.

- o The managed device may access the network in a way that dynamically assigns it an IP address and it doesn't register its assigned IP addressed to a mapping service.

- o The managed device may be configured in "stealth mode" and thus doesn't have any open ports.

- o The managed device may be deployed behind a firewall that doesn't allow SSH access to the internal network.

- o The managed device may be deployed behind a firewall that implements network address translation (NAT) for all internal network IP addresses.

- o The operator may prefer to have managed devices initiate management connections believing it is easier to secure one open-port in the data center than to have an open port on each managed device in the network.

One key benefit of using SSH as the transport protocol is its ability to multiplex an unspecified number of independently flow-controlled TCP sessions [RFC4254]. This is valuable as the managed device only needs to be configured to initiate a single Reverse SSH connection regardless the number of TCP-based protocols the application wishes to support. For instance, the application may "pin up" a channel for each distinct type of asynchronous notification the managed device supports (logs, traps, backups, etc.) and dynamically open/close channels as needed by its runtime.

4. The Reverse SSH Protocol

The NETCONF server's perspective

- The NETCONF server initiates a TCP connection to the NETCONF client on the IANA-assigned Reverse SSH port XXX.

- Immediately after the TCP session starts, the NETCONF server starts the SSH server protocol using the accepted TCP connection. That is, the NETCONF Server sends its SSH host key during the SSH key exchange.

The NETCONF client's perspective

- The NETCONF client listens for TCP connections on the IANA-assigned SSH port XXX.

- The NETCONF client accepts an incoming TCP connection and immediately starts the SSH client protocol. That is, the NETCONF client will need to authenticate its peer's SSH host key during the SSH key exchange.

This document updates the SSH Transport Layer Protocol [RFC4253] only by removing the restriction in Section 4 (Connection Setup) of [RFC4252] that the SSH Client must initiate the transport connection. Security implications related to this change are discussed in the Security Considerations (Section 7) section.

For first-time connections, in order for the NETCONF client to authenticate the NETCONF server, a public host key algorithm that certifies the the NETCONF server's identity and host-key SHOULD be used. Examples of suitable public host key algorithms are the x509v3-* algorithms defined in [RFC6187] and the the hmac-* algorithms defined in the The hmac-* Public Key Algorithms (Section 5) section below.

5. The hmac-* Public Key Algorithms

This section defines a family of public host key algorithms that can be used to both identify the SSH server and enable its host key to be automatically authenticated.

The algorithms presented in this section rely on a symmetric HMAC key to convey trust. This is in contrast to the PKI based authentication model used by the x.509 based public host key algorithms ([RFC6187]). An HMAC key enables Reverse SSH to be used in deployments where it's not possible for a x.509 Certificate Authority to sign the managed device's certificate in time, as it only requires a password to be provided.

The HMAC-based public host key algorithms defined in this specification mirror those defined in [RFC6187]. These host-keys are to be treated the same way as in [RFC6187], except that the peer authenticates the host key via an HMAC, instead of PKIX. The algorithms defined by this specification are:

Algorithm
hmac-ssh-dss
hmac-ssh-rsa
hmac-rsa2048-sha256
hmac-ecdsa-sha2-*

Regardless of which underlying host key is used, the format of the hmac-* based public key is as follows:

```
+---------------------+
|   string server-id  |
|   string host-key   |
|   string hmac       |
+---------------------+
```

The "server-id" field encodes a user-configured unique identifier for the SSH Server, or its Serial Number if none provided. This field is necessary as the SSH client MAY not otherwise be identifiable. For instance, the SSH server may be "calling home" for the first time or have a dynamically assigned address (DHCP, NAT, etc.).

The "host-key" field is the SSH Server's corresponding SSH host key. For instance, if the "hmac-ssh-rsa" public key was negotiated during key exchange, this field would encode the "ssh-rsa" host key.

The "hmac" field is the value produced using the MAC algorithm negotiated during key exchange over the selected host key and a user-configured HMAC key. [[RFC2104]]

6. Device Configuration

For devices supporting NETCONF, this section defines a YANG [RFC6020] module to configure Reverse SSH on the device. For devices that do not support NETCONF, this section illustrates what its configuration data model SHOULD include.

This YANG module enables a NETCONF client to generically manage a NETCONF server's Reverse SSH configuration. Key aspects of this YANG module include support for more than one application, more than one server per application, and a reconnection strategy.

This RFC does not attempt to define any strategy for how an initial deployment might obtain its bootstrapping "call home" configuration, as defined by this YANG module. That said, implementations may consider fetching configuration from a server identified via the DHCP protocol or loading it off a USB drive plugged into the device before being powered on.

Configuration Example


```
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <reverse-ssh xmlns="urn:ietf:params:xml:ns:yang:ietf-reverse-ssh">
      <applications>
         <application>
            <name>config-mgr</name>
            <description>
               This entry requests the device to periodically
               connect to the Configuration Manager application
            </description>
            <servers>
               <server>
                  <host>config-mgr1.acme.com</host>
                  <port>7022</port>
               </server>
               <server>
                  <host>config-mgr2.acme.com</host>
                  <port>7022</port>
               </server>
            </servers>
            <periodic-connection>
               <timeout-mins>5</timeout-mins>
               <linger-secs>20</linger-secs>
            </periodic-connection>
            <host-keys>
               <host-key>
                  <name>ssh_host_key_cert</name>
               </host-key>
```

```
            <host-key>
               <name>ssh_host_key_cert2</name>
            </host-key>
         </host-keys>
         <keep-alive-strategy>
            <interval-secs>5</interval-secs>
            <count-max>3</count-max>
         </keep-alive-strategy>
         <reconnect-strategy>
            <start-with>last-connected</start-with>
            <interval-secs>10</interval-secs>
            <count-max>4</count-max>
         </reconnect-strategy>
      </application>
      <application>
         <name>log-monitor</name>
         <description>
            This entry requests the device to mantain a
            persistent connection to the Log Monitoring
            application
         </description>
         <servers>
            <server>
               <host>log-mon1.acme.com</host>
               <port>7514</port>
            </server>
            <server>
               <host>log-monitor2.acme.com</host>
               <port>7514</port>
            </server>
         </servers>
         <persistent-connection/>
         <host-keys>
            <host-key>
               <name>ssh_host_key_hmac</name>
            </host-key>
         </host-keys>
         <keep-alive-strategy>
            <interval-secs>5</interval-secs>
            <count-max>3</count-max>
         </keep-alive-strategy>
         <reconnect-strategy>
            <start-with>last-connected</start-with>
            <interval-secs>10</interval-secs>
            <count-max>4</count-max>
         </reconnect-strategy>
      </application>
   </applications>
```

```
      </reverse-ssh>
   </config>
```

The YANG Module

```
   module ietf-reverse-ssh {

      namespace "urn:ietf:params:xml:ns:yang:ietf-reverse-ssh";

      prefix "rssh";

      import ietf-inet-types { prefix inet; }

      organization
        "IETF NETCONF (Network Configuration Protocol) Working Group";

      contact
        "WG Web:   <http://tools.ietf.org/wg/netconf/>
         WG List:  <mailto:netconf@ietf.org>

         WG Chair: Bert Wijnen
                   <mailto:bertietf@bwijnen.net>

         WG Chair: Mehmet Ersue
                   <mailto:mehmet.ersue@nsn.com>

         Editor: Kent Watsen
                 <mailto:kwatsen@juniper.net>";


      revision 2013-06-18 {
          description "Initial conception";
          reference "RFC XXXX: Reverse SSH";
      }
      // RFC Ed.: replace XXXX with actual
      // RFC number and remove this note


      container reverse-ssh {
          container applications {
              description
                  "All the application that the device
                   initiates Reverse SSH connections to";
              list application {
                  key name;
                  min-elements 1;
                  leaf name {
                      mandatory true;
```

```
                type string {
                    length 1..32;
                }
                description
                   "The name of the application the device is
                    connecting to";
            }
            leaf description {
                type string;
                description
                  "An optional description for the application";
            }
            container servers {
                description
                    "An ordered listing of the application's
                     servers that the device should attempt
                     connecting to.";
                list server {
                    key host;
                    min-elements 1;
                    ordered-by user;
                    leaf host {
                        mandatory true;
                        type inet:host;
                        description
                            "IP address or domain-name for
                             the server";
                    }
                    leaf port {
                        type inet:port-number;
                        description
                            "The IP port for this server.
                             The device will use the
                             IANA-assigned port if not
                             specified.";
                    }
                }
            }

            choice connection-type {
                description "Indicates the application's
                             preference for how the device's
                             connection is maintained.";
                default persistent-connection;
                leaf persistent-connection {
                    type empty;
                }
                container periodic-connection {
```

```
                leaf timeout-mins {
                    type uint8;
                    default 5;
                    units minutes;
                    description
                       "The maximum amount of unconnected
                        time the device will wait until
                        establishing a connection to the
                        applications again to send it.
                        The device may establish a
                        connection before this time if
                        it has data it needs to send to
                        the device.";
                }
                leaf linger-secs {
                    type uint8;
                    default 30;
                    units seconds;
                    description
                       "The amount of time the device should
                        wait after last receiving data from
                        or sending data to the device before
                        closing its connection to the app.";
                }
            }
        }
        container host-keys {
            description
                "An ordered listing of the SSH host keys the
                 device should advertise to the application.";
            list host-key {
                key name;
                min-elements 1;
                ordered-by user;
                leaf name {
                    mandatory true;
                    type string {
                        length 1..64;
                    }
                    description
                       "The name of a host key the device
                        should advertise during the SSH
                        key exchange.";
                }
            }
        }
        container keep-alive-strategy {
            leaf interval-secs {
```

```
                 type uint8;
                 units seconds;
                 default 15;
                 description
                   "Sets a timeout interval in seconds after
                    which if no data has been received from
                    the client, a message will be sent to
                    request a response from the SSH client.
                    A value of '0' indicates that no messages
                    should be sent.";
             }
             leaf count-max {
                 type uint8;
                 default 3;
                 description
                   "Sets the number of keep alive messages
                    that may be sent without receiving any
                    response from the SSH client before
                    assuming the SSH client is no longer
                    alive.  If this threshold is reached
                    the device will disconnect the SSH
                    session.  The keep alive interval timer
                    is reset after each transmission.  Thus,
                    an unresponsive SSH client will be
                    disconnected after approximately
                    'count-max * interval-secs' seconds.";
             }
         }
         container reconnect-strategy {
             leaf start-with {
                 default first-listed;
                 type enumeration {
                     enum first-listed;
                     enum last-connected;
                 }
             }
             leaf interval-secs {
                 type uint8;
                 units seconds;
                 default 5;
                 description
                   "time delay between connection attempts";
             }
             leaf count-max {
                 type uint8;
                 default 3;
                 description
                   "num times try to connect to a server";
```

```
                      }
                  }
              }
          }
      }
   }
```

7. Security Considerations

This RFC deviates from standard SSH protocol usage by allowing the SSH server to initiate the TCP connection. This conflicts with section 4 of the SSH Transport Layer Protocol RFC [RFC4253], which states "The client initiates the connection". However this statement is made without rationalization and it's not clear how it impacts the security of the protocol, so this section analyzes the security offered by the having the client initiate the connection.

First, assuming the SSH server is not using a public host key algorithm that certifies its identity, the security of the protocol doesn't seem to be sensitive to which peer initiates the connection. That is, it is still the case that reliable distribution of host keys (or their fingerprints) should occur prior to first connection and that verification for subsequent connections happens by comparing the host keys in locally cached database. It does not seem to matter if the SSH Server's host name is derived from user-input or extracted from the TCP layer, potentially via a reverse-DNS lookup. Once the host name-to-key association is stored in a local database, no man-in-the-middle attack is possible due to the attacker being unable to guess the real SSH server's private key (Section 9.3.4 (Man-in-th-middle) of [RFC4251]).

That said, this RFC recommends implementations use a public host key algorithm that certifies the SSH server's identity. The identity can be any unique identifier, such as a device's serial number or a deployment-specific value. If this recommendation is followed, then no information from the TCP layer would be needed to lookup the device in a local database and therefore the directionality of the TCP layer is clearly inconsequential.

The SSH protocol negotiates which algorithms it will use during key exchange (Section 7.1 (Algortihm Negotition) in [RFC4253]). The algorithm selected is essentially the first compatible algorithm listed by the SSH client that is also listed by the SSH server. For a network management application, there may be a need to advertise a large number of algorithms to be compatible with the various devices it manages. It is RECOMMENDED that the SSH client orders its list of public host key algorithms such that all the certifiable public host key algorithms are listed first. Additionally, when possible, SSH

servers SHOULD only list certifiable public host key algorithms. Note that since the SSH server would have to be configured to know which IP address it needs to connect to, it is expected that it will also be configured to know which host key algorithm to use for the particular application, and hence only needs to list just that one public host key algorithm.

This RFC suggests implementations can use a device's serial number as a form of identity. A potential concern with using a serial number is that the SSH protocol passes the SSH server's host-key in the clear and many times serial numbers encode revealing information about the device, such as what kind of device it is and when it was manufactured. While there is little security in trying to hide this information from an attacker, it is understood that some deployments may want to keep this information private. If this is a concern, deployments MAY consider using instead a hash of the device's serial number or an application-specified unique identifier.

The HMAC-* family of public host key algorithms defined in this RFC take a hmac-key. The length of the hmac-key SHOULD NOT be less than the output length of the associated hash function, as discussed in Section 3 (Keys) in [RFC2104]. The associated hash function for each public host key algorithm is as follows:

Algorithm	Hash Function(s)
hmac-ssh-dss	SHA-1
hmac-ssh-rsa	SHA-1
hmac-rsa2048-sha256	SHA-256
hmac-ecdsa-sha2-*	SHA-256, SHA-384, or SHA-512

Note: for the Elliptical Curve algorithms, the hash function selection is defined by Section 6.2.1 in [RFC5656].

The output length for each of these hash functions is as follows:

Hash Function	Output Length (bytes)
SHA-1	20
SHA-256	32
SHA-384	48
SHA-512	64

The hmac-* public host key algorithms require the application consume the <server-id> field without being able to first verify that it is the value the managed device sent. The application must use the server-id value to lookup the managed device's record in a local datastore in order to obtain the HMAC-key needed to authenticate the HMAC. The application must be sure to process the server-id carefully as it may have been purposely encoded to illicit unexpected behaviour.

An attacker could DoS the application using valid "server-id" values, forcing the application to perform computationally expensive operations, only to deduce that the attacker doesn't posses a valid key. This is no different than any secured service and all common precautions apply (e.g. blacklisting the source address after a set number of unsuccessful login attempts).

8. IANA Considerations

Consistent with Section 8 of [[RFC4251]] and Section 4.6 of [[RFC4250]], this document makes the following registrations in the Public Key Algorithm Names registry:

- The SSH public key algorithm "hmac-ssh-dss".

- The SSH public key algorithm "hmac-ssh-rsa".

- The SSH public key algorithm "hmac-rsa2048-sha256".

- The family of SSH public key algorithm names beginning with "hmac-ecdsa-sha2-" and not containing the at-sign ('@').

This document requests that IANA assigns a TCP port number in the "Registered Port Numbers" range with the service name "reverse-ssh". This port will be the default port for the Reverse SSH protocol and will be used when the NETCONF server needs to initiate a connection to a NETCONF client using SSH. Below is the registration template following the rules in [RFC6335].

```
Service Name:           reverse-ssh
Transport Protocol(s):  TCP
Assignee:               IESG <iesg@ietf.org>
Contact:                IETF Chair <chair@ietf.org>
Description:            Reverse SSH (call home)
Reference:              RFC XXXX
Port Number:            YYYY
```

9. Normative References

[RFC2104] Krawczyk, H., Bellare, M., and R. Centti, "HMAC: Keyed-Hashing for Message Authentication ", RFC 2104, February 1997.

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

[RFC4250] Lehtinen, S. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Assigned Numbers ", RFC 4250, December 2005.

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

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

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

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

[RFC5656] Stebila, D. and J. Green, "Elliptic Curve Algorithm Integration in the Secure Shell Transport Layer ", RFC 5656, December 2009.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF) ", RFC 6020, October 2010.

[RFC6187] Igoe, K. and D. Stebila, "X.509v3 Certificates for Secure Shell Authentication ", RFC 6187, March 2011.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "NETCONF Configuration Protocol", RFC 6241, June 2011.

[RFC6242] Wasserman, M., Ed., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011.

[RFC6335] Cotton, M., Ed., Eggert, L., Ed., Touch, J., Ed., Westerlund, M., Ed., and S. Cheshire, Ed., "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", RFC 6335, August 2011.

Author's Address

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net