NETCONF Working Group
Internet-Draft
Updates: 4253 (if approved)
Intended status: Standards Track
Expires: May 05, 2014

K. Watsen
Juniper Networks

November 2013

Reverse Secure Shell (Reverse SSH)
draft-ietf-netconf-reverse-ssh-02

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 YYYY 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

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on May 05, 2014.

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

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 YYYY 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].

2.1. Applicability Statement

The techniques described in this document are suitable for network management scenarios such as the ones described in section 3. However, these techniques MUST only be used for a NETCONF server to initiate a connection to a NETCONF client, as described in this document.

The reason for this restriction is that different protocols have different security assumptions. The NETCONF over SSH specification requires NETCONF clients and servers to verify the identity of the other party before starting the NETCONF protocol. This contrasts with the base SSH protocol, which does not require programmatic verification of the other party. In such circumstances, allowing the SSH server to contact the SSH client would open new vulnerabilities. Therefore, any use of Reverse SSH for purposes other than NETCONF will need a thorough, contextual security analysis.

2.2. Update to RFC 4253

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.

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 network element to be the SSH server.

Reverse SSH enables the network element 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:

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

- The network element 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.

- The network element may be configured in "stealth mode" and thus doesn't have any open ports for the management system to connect to.

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

- The network element may be deployed behind a firewall that implements network address translation (NAT) for all internal network IP addresses, thus complicating the ability for a management system to connect to it.

- The operator may prefer to have network elements initiate management connections believing it is easier to secure one open-port in the data center than to have an open port on each network element 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 network element only needs to be configured to initiate a single Reverse SSH connection to the management system, regardless the number of TCP-based protocols the management system wishes to support. For instance, in addition to having a SSH channel for NETCONF, management system may "pin up" channels for Syslog, SNMP, or file-transfers.

4. The Reverse SSH Protocol

The NETCONF server's perspective (e.g. the network element)

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

- The TCP connection is accecpted and a TCP session is established.

- Using this TCP connection, the NETCONF server immediately starts the SSH Server protocol. That is, the next message sent on the TCP stream is SSH's Protocol Version Exchange message (section 4.2, [RFC4253]).

The NETCONF client's perspective (e.g. the management system)

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

- The NETCONF client accepts an incoming TCP connection and a TCP session is established.

- Using this TCP connection, the NETCONF client immediately starts the SSH Client protocol, starting with sending the SSH's Protocol Version Exchange message (section 4.2, [RFC4253]).

5. SSH Server Identification and Verification

When the management system accepts a new incoming connection, it needs to authenticate the remote peer. Ultimately, this comes down to identifying the peer and verifying its SSH host key.

Due to Reverse SSH having the network element initiate the TCP connection, the first data the management system has to identify it with is the source IP address of the TCP connection. But this approach only works in networks that only use static addresses.

To support network-elements having dynamically-assigned IP addresses or deployed behind gateways that translate their IP address (e.g. NAT), the management system MAY identify the device using its SSH host key. For instance, a fingerprint of the network element's host key could be used as an identifier since the probability of collision is acceptibly low. But this solution requires the management system to be configured with each device's host key each time it changes.

Yet another option for identifying the network element if for it's host key to encode its identity, such as if it were a certificate. This option enables the host key to change over time, but brings the next issue of how the mangement element can verify the network element's host key is authentice.

The security of SSH is anchored in the ability for the SSH client to verify the SSH server's hostkey. Typically this is done by comparing the host key presented by the SSH server with one that was previously configured on the SSH client, looking it up in a local database using the identity of the SSH client as the lookup key. Nothing changes regarding this requirement due to the direction reversal of the underlying TCP connection. To ensure security, the management system MUST verify the network element's SSH host key each time a SSH session is established.

However, configuring distinct host keys on the management system doesn't scale well, which is an important consideration to a network management system. A more scalable strategy is to have the network element's host key signed by a common trusted key, such as a certificate authority. Thus, the mangement system only needs to trust a single public key, which vouches for the authenticity of the network element's specific public key.

Since both the identification and verification issues are addressed using certificates, this draft RECOMMENDS network elements use a host key that can encode a unique (e.g. its serial number) and be signed by a common trust anchor (e.g. a certificate authority). Examples of suitable public host keys are the X.509v3 keys defined in defined in [RFC6187].

6. Device Configuration

This section defines a YANG [RFC6020] module to configure Reverse SSH on the device. 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.

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>
            <host-keys>
               <host-key>
                  <name>ssh_host_key_cert</name>
               </host-key>
               <host-key>
                  <name>ssh_host_key_cert2</name>
               </host-key>
            </host-keys>
            <connection-type>
                <periodic>
                    <timeout-mins>8</timeout-mins>
```

```
                    <linger-secs>10</linger-secs>
                </periodic>
            </connection-type>
            <reconnect-strategy>
               <start-with>last-connected</start-with>
               <interval-secs>10</interval-secs>
               <count-max>4</count-max>
            </reconnect-strategy>
            <keep-alive-strategy>
               <interval-secs>5</interval-secs>
               <count-max>3</count-max>
            </keep-alive-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>
            <host-keys>
               <host-key>
                  <name>ssh_host_key_hmac</name>
               </host-key>
            </host-keys>
            <connection-type>
                <persistent/>
            </connection-type>
            <reconnect-strategy>
               <start-with>last-connected</start-with>
               <interval-secs>10</interval-secs>
               <count-max>4</count-max>
            </reconnect-strategy>
            <keep-alive-strategy>
               <interval-secs>5</interval-secs>
               <count-max>3</count-max>
            </keep-alive-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..64;
                }
                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.";
                   }
                }
             }
             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 connection-type {
              description "Indicates the application's
                       preference for how the device's
                       connection is maintained.";
              choice connection-type {
                 default persistent-connection;
                 case persistent-connection {
                    leaf persistent {
                       type empty;
                    }
                 }
                 case periodic-connection {
                    container periodic {
                       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 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";
              }
           }
           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.";
                    }
                }
            }
        }
    }
}
```

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 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. The SSH client SHOULD order 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.

An attacker could DoS the application by having it to perform computationally expensive operations, before deducing 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

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. Acknowledgements

The author would like to thank for following for lively discussions on list and in the halls (ordered by last name): Andy Bierman, Martin Bjorklund, Mehmet Ersue, Wes Hardaker, Stephen Hanna, David Harrington, Jeffrey Hutzelman, Mouse, Russ Mundy, Tom Petch, Peter Saint-Andre, Joe Touch, Sean Turner, Bert Wijnen.

10. Normative References

[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.

[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.

Appendix A. Change Log

A.1. 01 to 02

Added Applicability Statement

Removed references to ZeroConf / ZeroTouch

Clarified the protocol section

Added a section for identification and verification

A.2. 00 to 01

removed the hmac-* family of algorithms

Author's Address

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net