Zero Touch Provisioning for NETCONF Call Home (ZeroTouch)  
draft-kwatsen-netconf-zerotouch-01

Abstract

This draft presents a technique for how to establish a secure NETCONF connection between a newly deployed networking device, configured with just its factory default settings, and the new owner’s Network Management System (NMS).

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 August 05, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

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

The solution presented herein is designed to support the NETCONF configuration protocol [RFC6241]. This is achieved by leveraging the recently standardized call home mechanisms for SSH [REVERSE-SSH] and TLS [RFC5539bis].
A fundamental business requirement is to reduce operational costs where possible. Deploying new devices is typically one of the largest costs in running the network, as sending trained specialists to each site is both cost prohibitive and doesn’t scale.

Both networking vendors and standard bodies have tried to address this issue, with varying levels of success. For instance, the Broadband Forum TR-069 specification [TR069] relies on DHCP for NMS discovery, but this only works in environments where the DHCP server can be configured, which isn’t the case when the device is connected to an ISP’s network. In another example, some network vendors have enabled their devices to load an initial configuration from removable storage media (e.g., a USB flash drive), but not all devices have such ports.

The solution presented herein attempts to address the evaluation criteria outlined by the draft "Configuring Security Parameters in Small Devices" [draft-hanna-zeroconf-seccfg-00]: security, flexibility, ease of use, and device cost. More specifically:

- **Security**
  
  Security is fundamental to any automated discovery solution, especially one that bootstraps the parameters used to secure a device’s connection to its NMS. Consistent with [RFC3365], security is a required aspect of ZeroTouch. Every ZeroTouch implementation is sure to be secure.

- **Flexibility**
  
  ZeroTouch is designed to support a wide variety of deployments, including cases where the device is connected to a network without administrative control of the local DHCP and DNS servers, where the device is connected to an untrusted network, or deployed behind a gateway that dynamically translates its network address (e.g., NAT). Special consideration is also provided for devices that are on a network with no public Internet access.

- **Ease of Use**
  
  Ultimately, the success of the solution depends on the ability for presumably non-technical personnel to be able to complete the installation by themselves. To this end, it is envisioned that installers only need to connect the device to a wired or wireless network and a power source and wait for the device to indicate success. ZeroTouch also attempts to not be overly complicated for NMS administrators.
For vendors of devices with already slim margins, such as consumer-oriented devices, a significant concern is the cost of goods. Fortunately, the solution presented in this draft requires minimal additional components. Additionally, the development effort doesn’t seem exorbitant, though that may vary by vendor and their circumstances.

3. Actors and Roles

3.1. Device

The device is the networking entity that initiates Zero Touch. Whenever a device boots with its factory default settings, it initiates ZeroTouch with the goal of finding a Configlet with which it can use to configure itself. Once a Configlet is found, the device initializes its configuration using that Configlet and then exits ZeroTouch. Since the Configlet configures the device to "call home" upon entering its normal operating mode, the device immediately begins trying to establish a reverse-SSH or reverse-TLS connection, as specified by the Configlet.

3.2. Configlet

A Configlet is an XML file, containing specific YANG-defined configuration, that has been signed by a trusted signer known to the device (e.g., the device’s manufacturer).

The Configlet data-model, defined by the YANG module in this document (see Section 3.2), is just enough to configure a local user account and either reverse-SSH or reverse-TLS. More specifically, this data-model is a subset of what’s defined in ietf-system and ietf-netconf-server YANG models. This focused data-model is consistent with the common use-case of having the NMS push a full configuration to a device when it calls home.

The signature on the Configlet is enveloped, meaning that the signature is contained inside the XML file itself. The signature block also contains the X.509 certificate of the Configlet Signer and its chain of trust.

Once a device authenticates the signature on a Configlet and matches the unique identifier (e.g., serial number) within the Configlet, it merges the configuration contained in the Configlet into its running datastore.
3.3. Configlet Signer

A Configlet Signer is the entity authorized by the device manufacturer to sign Configlets for its devices (note: this may be the device vendor itself).

A Configlet Signer MUST provide a user-facing service enabling the creation of a Configlet with user-specific deployment values, using the YANG schema defined in Section 3.2. This document does not specify what form this interface must take, so it is the Configlet Signer’s discretion if it is a web page, a REST API, or something else.

A Configlet Signer MUST ensure that the end-user is the rightful owner of the device containing the unique identifier value to be put into the Configlet. How a Configlet Signer ensures this is outside the scope of this document, but it is envisioned that the vendor would provide a secured interface for its trusted Configlet Signers to use.

A Configlet Signer MUST have an X.509 certificate with Key Usage capable of signing data (digitalSignature) and be signed by a certificate authority having a chain of trust leading to a trust anchor known to the devices loading its Configlets. The Configlet Signer MUST possess all intermediate certificates leading to its trust anchor.

When a Configlet Signer signs a Configlet, it attaches both the signature and the chain of X.509 certificates, including its own, but not necessarily including the trust anchor’s certificate. This chain of certificates is needed so a device can validate a Configlet using only the Configlet Signer trust anchors known to it.

A Configlet Signer does not need to retain the Configlet after signing it; it is expected that either the end-user or the Configlet Signer will convey the signer Configlet to a Configuration Server where it will be hosted.

3.4. Configuration Server

A Configuration Server is the entity hosting configurations that can be downloaded over a network. Configuration Servers are known to devices in the form of a URI, to which a device appends the fingerprint of its entity certificate (see Section 4.1 for details). For instance, if the URI were:
https://example.com?id=
scp://user@zerotouch.example.com/configlets/
ftp://example.com/zerotouch/configlets/

then the device would try to access:

https://example.com?id=<fingerprint>
scp://user@zerotouch.example.com/configlets/<fingerprint>
ftp://example.com/zerotouch/configlets/<fingerprint>

where the fingerprint is generated using the SHA-256 algorithm over
the device’s entity certificate. For instance, using OpenSSL’s
command line tool: ‘openssl dgst -sha256 <entity certificate>’ (see
Section 4.1).

The Configuration Server is expected to be able to map the
fingerprint to a device-specific unique identifier (e.g., serial
number), which is the value contained in the Configlet it is hosting.
How the Configuration Server does this mapping is outside the scope
of this document, but it is envisioned that the vendor would provide
a secured interface for its trusted Configuration Servers to use.

Configuration Servers do not need to use encryption, since the
Configlets themselves are immutable to tampering, due to being
signed. However, for confidentiality reasons, it is RECOMMENDED to
use encryption, so adversaries cannot see the Configlet’s otherwise
clear-text content, from which they can learn some details about the
device’s internal deployment.

If a Configuration Server uses X.509-based encryption, then its X.509
certificate MUST have a chain of trust to a trust anchor known to
devices. The Configuration Server MUST possess all the intermediate
certificates leading to a trust anchor.

When a Configuration Server negotiates encryption with the device, it
provides the chain of X.509 certificates, including its own, but not
necessarily including the trust anchor’s certificate. Devices need
the chain of certificates to be passed so they can validate the
server using only a minimal list of Configuration Server trust
anchors.

Configuration Server’s SHOULD automatically expire Configlets after
some user-specified amount of time.

In order to facilitate troubleshooting and auditing, the
Configuration Server SHOULD record into a log a record of the various
Configlet download requests. This draft does not define what
information should be kept or for how long.
3.5. Network Management System (NMS)

The NMS is the ultimate destination of ZeroTouch for a device. It is the NMS’s network address configured in the Configlet. The device will connect to the NMS using either a reverse-SSH or reverse-TLS, as configured by the Configlet loaded.

In order to authenticate the device, the NMS MUST possess the X.509 certificate for the trust anchor leading to the device’s entity certificate. The NMS uses this certificate to validate the server-certificate the device presents during SSH or TLS transport negotiation.

The NMS SHOULD also validate that the unique identifier (e.g., serial number), within the "Common Name" field of the device’s X.509 certificate, is an identity that the NMS is expecting, and not another device having the same device type. In order for the NMS to know the unique identifiers for devices shipped directly to their destinations, it may be necessary for the device manufacturer to provide the unique identifiers along with other shipping or billing information. This draft not specify a format for this information exchange.

In addition to authenticating the device, the NMS must also authenticate itself to the device. How this is done is protocol specific. For reverse-SSH, the NMS needs to know the information configured on the device by the Configlet it loaded, specifically the name of a local user account and the necessary credentials configured for the account (i.e., password or the private-half of a SSH host key). For reverse-TLS, the NMS must present a client certificate. Presumably the NMS has been configured with the information used when the Configlet was created.

4. Device Boot Sequence

4.1. Precondition

Devices supporting ZeroTouch MUST support either reverse-SSH or reverse-TLS, and MAY support both. In either case, the device MUST present an X.509-based certificate encoding a unique device identifier (e.g. serial number) and a public key, both signed by a trusted certificate authority. This certificate is the "entity certificate" in the diagram below. This certificate is needed in order for the NMS to positively authenticate a device. For reverse-SSH, this certificate requirement constrains the SSH host key algorithms the device is allowed to to use to those defined in [RFC6187].
The unique identifier MUST be something that is both known to the
device and easily tracked through labels affixed to the device as
well as the box it is packaged in. A device’s serial number is
commonly treated this way and would be suitable for this purpose.

The device MUST possess the private key matching the public key
encoded in the entity certificate. Ideally, the private key SHOULD
be generated and protected by a tamper-resistant cryptographic
processor, as this provides the greatest assurance that the private
key is known to no one, including the device’s manufacturer.

The entity certificate MUST be signed by a certificate authority
having a chain of trust to a well-known trust anchor. The device
MUST also possess the X.509 certificates for any intermediate CAs
leading to the trust anchor. During SSH or TLS transport setup, the
device will send both its entity and any intermediate certificates so
the NMS can verify the certificate path using only the well-known
trust anchor.

Since the entity certificate is to be used for SSH and TLS
connections, its Key Usage, if set, SHOULD have the digitalSignature,
keyEncipherment, and keyAgreement bits set.

In order for a device to know where it can obtain a Configlet, it
MUST have two sets of URIs, identifying resources where it can obtain
Configlets. One set contains secure schemes (e.g. https://, scp://)
and the other contains insecure schemes (e.g., http://, ftp://).
These URIs point to the “Configuration Servers” in the diagram below.

In order for a device to use a URI with a secure scheme, devices MUST
possess a trust anchor certificate that it can use to authenticate
the Configuration Server with. As each Configuration Server may use
a different trust anchor, this generalizes to a list of Configuration
Server trust anchor certificates. These trust anchors MAY include
broadly recognized certificate authorities, such as the certificates
packaged with web browsers.

In order for a device to authenticate Configlets, it MUST have a
trust anchor for the CA that signed the Configlet. The CA used to
sign Configlets is called a "Configlet Signer" in the diagram below.
In order to enable Configlets to be signed by different CAs, the
device MAY have either a list of trust anchors, or a single trust
anchor that delegates Configlet signing trust to other CAs. The
diagram below shows a list of Configlet Signer trust anchors only
because it is the more flexible option.

Devices SHOULD ensure that all its trust anchor certificates,
including those for the Configlet Signer and Configuration Server,
are protected from external modification. It is for this reason that the diagram below shows them in immutable storage. However, it may be necessary to update these certificates over time (e.g., the vendor wants to delegate trust to a new CA). Therefore, devices MAY update these trust anchors when needed through a verifiable process, such as a software upgrade using signed software images.

Devices SHOULD ensure that the certificates for its trust anchors are protected from external modification, specifically the Configlet Signer and Configuration Server X.509 certificates. It is for this reason that the diagram below shows them in immutable storage. The certificates for the device’s trust anchors MAY be updated along with a standard software image upgrade.

Device State Precondition

4.2. Runtime Operation

Whenever a device boots with its factory default settings, it initiates ZeroTouch with the goal of finding a configuration with which it can use to configure itself.
The device MUST first initialize its networking as per its default factory configuration. This SHOULD result in the dynamic assignment of an IP address, subnet or prefix, gateway, and a DNS server.

While initializing its networking, the device MAY receive some additional URIs for where a software image or configuration can be downloaded. This draft does not define how such URIs MAY be exchanged, for instance, using DHCP options.

If, while initializing its networking, the device receives software image URIs, it MAY download and install the software image only if the image is protected from modification (e.g., the image is signed) and the device is able to verify its integrity. The device SHOULD try URIs with secure schemes before URIs with insecure schemes (e.g., scp:// before ftp://). If the device needs to reboot to activate the new software image, it MUST do so with its default factory configuration set so that ZeroTouch will run again when the device comes back up.

If, while initializing its networking, the device receives configuration URIs, each URI SHOULD be appended to one of the device’s two sets of Configuration Server URIs, depending on if the URI’s scheme is secure or not. URIs added this way MUST remain distinguishable from those URIs the device was shipped with, for reasons discussed in Section 4.2.

Before trying any of the Configuration Server URIs, the device SHOULD first try to load a configuration through local means that assert physical presence. For instance, a removable USB flash drive or near-field communication mechanism. Configurations obtained through an assertion of physical presence do not have to be signed or contain the device’s unique identifier (e.g., serial number). If a Configlet is found, the device MUST use it without trying any of the Configuration Server URIs.

If a configuration was not found via physical presence, the device then iterates over its two sets of Configuration Server URIs. The device MUST first try all the URIs from the set having secure schemes before trying any of the URIs from the set having insecure schemes. For each URI, until a usable configuration is found and successfully loaded, the device attempts to download a configuration from the URI. If the URI uses a secure scheme (e.g., https), the device MUST validate the Configuration Server’s certificate using one of its Configuration Server trust anchors. If the device is unable to verify the server’s certificate, the device MUST skip that URI. If the device reaches the end of all its URIs without finding a configuration, it MAY continue its normal boot sequence using its factory default configuration.
When the device is accessing a Configuration Server URI that it was shipped with (i.e., not discovered while initializing its networking, it MUST do so by appending the fingerprint of its entity certificate to the URI’s string, as described in the Section 3.4. For URIs discovered while initializing its networking, the device MAY try both the raw URI as well as the permutation of it using its fingerprint.

The Configuration Server’s response MAY be anything allowed by the given URI’s scheme. For instance, if the scheme is HTTP-based, the Configuration Server MAY return an HTTP redirect. In this way, a vendor’s Configuration Server service may allow the device owner to redirect the device to a Configuration Server running in their network.

If the Configuration Server returns a Configlet, the device MUST first verify it before use. Configlet verification entails both verifying the Configlet’s signature using the device’s list of Configlet Signer trust anchors, and also verifying that the unique identifier (e.g., serial number) within the Configlet matches the device’s unique identifier.

Once a Configlet is authenticated, the device merges the Configlet’s contents into its running configuration and then exits ZeroTouch. Since the Configlet configures the device to "call home," upon entering its normal operating mode, the device immediately begins trying to establish a reverse-SSH or reverse-TLS connection, as specified by the Configlet.

If configured to establish a reverse-SSH connection, the device MUST use its entity and associated intermediate X.509 certificates as its host key per RFC 6187 [RFC6187]. If configured to use reverse-TLS, the device MUST use its entity and associated intermediate X.509 certificates as its server-side certificate for the TLS connection.

In order to facilitate troubleshooting, the device SHOULD record into a log information relating to its stepping through the ZeroTouch sequence of steps. This draft does not define any specific log messages, for instance, for Syslog or SNMP.

ZeroTouch Sequence Diagram
5. Configlets

5.1. Data Model

The Configlet’s data is modeled after the data models provided by draft-ietf-netmod-system-mgmt and draft-kwatsen-netconf-server. These data models are used to configure a local user account and either reverse-SSH or reverse-TLS. Networking is not included in the Configlet data model as it is expected that the device will be assigned a dynamic address by the network and that it will use this address both when connecting to a Configuration Server and later the NMS.

From draft-ietf-netmod-system-mgmt, the data model for user authentication has the following structure:

```
  +--rw system
    +--rw authentication
      +--rw user-authentication-order*  identityref
      +--rw user* [name]
        +--rw name         string
        +--rw password?    crypt-hash
        +--rw ssh-key* [name]
          +--rw name         string
          +--rw algorithm    string
          +--rw key-data     binary
```
From `draft-kwatsen-netconf-server`, the data model for reverse-SSH has the following structure:

```
++-rw netconf
  ++-rw ssh {ssh}?
    ++-rw listen {inbound-ssh}?
      ++-rw (one-or-many)?
        +--:(one-port)
        |  ++-rw port? inet:port-number
        +--:(many-ports)
        |  ++-rw interface* [address]
        |    ++-rw address inet:ip-address
        |    ++-rw port? inet:port-number
    ++-rw call-home {outbound-ssh}?
  ++-rw applications
    ++-rw application* [name]
      ++-rw name string
      ++-rw description? string
      ++-rw servers
        ++-rw server* [address]
        |  ++-rw address inet:host
        |  ++-rw port? inet:port-number
      ++-rw connection-type
        ++-rw (connection-type)?
        |  ++-:(persistent-connection)
        |    ++-rw persistent
        |    ++-rw keep-alives
        |    |  ++-rw interval-secs? uint8
        |    |  ++-rw count-max? uint8
        |    ++-:(periodic-connection)
        |    ++-rw periodic
        |    |  ++-rw timeout-mins? uint8
        |    |  ++-rw linger-secs? uint8
    ++-rw reconnect-strategy
      ++-rw start-with? enumeration
      ++-rw interval-secs? uint8
      ++-rw count-max? uint8
    ++-rw host-keys
      ++-rw host-key* [name]
      ++-rw name string
```

Also from `draft-kwatsen-netconf-server`, the data model for reverse-TLS has the following structure:
+-rw netconf
  +-rw tls {tls}?
    +-rw listen {inbound-tls}?
      +-rw (one-or-many)?
        +-:(one-port)
        |  +-rw port?        inet:port-number
        +-:(many-ports)
        |  +-rw interface* [address]
        |      +-rw address    inet:ip-address
        |      +-rw port?      inet:port-number
    +-rw call-home {outbound-tls}?
      +-rw applications
        +-rw application* [name]
          +-rw name                  string
          +-rw description?          string
        +-rw servers
          |  +-rw server* [address]
          |     +-rw address    inet:host
          |     +-rw port?      inet:port-number
        +-rw connection-type
          |  +-rw (connection-type)?
          |     +-:(persistent-connection)
          |        |  +-rw persistent
          |        |     +-rw keep-alives
          |        |        +-rw interval-secs?   uint8
          |        |     +-rw count-max?       uint8
          |     +-:(periodic-connection)
          |        +-rw periodic
          |           +-rw timeout-mins?   uint8
          |           +-rw linger-secs?    uint8
        +-rw reconnect-strategy
          |  +-rw start-with?      enumeration
          +-rw interval-secs?   uint8
          +-rw count-max?       uint8
    +-rw cert-maps {tls-map-certificates}?
      +-rw cert-to-name* [id]
        |  +-rw id             uint32
        |  +-rw fingerprint    x509c2n:tls-fingerprint
        |  +-rw map-type       identityref
        |  +-rw name           string
    +-rw psk-maps {tls-map-pre-shared-keys}?
      +-rw psk-map* [psk-identity]
        |  +-rw psk-identity    string
        |  +-rw user-name       nacm:user-name-type
        |  +-rw not-valid-before?  yang:date-and-time
        |  +-rw not-valid-after?  yang:date-and-time
        |  +-rw key             yang:hex-string
5.2. Signature

Configlets obtained over the network MUST be signed using the W3C standard "XML Signature Syntax and Processing" [XMLSIG]. The entire contents of the Configlet MUST be signed. The signature block must also include the Configlet Signer's certificate and any intermediate certificates leading to a Configlet Signer trust anchor.

A signed Configlet example is in the Appendix.

5.3. YANG Module

Following is the YANG module for the Configlet:

module ietf-netconf-zerotouch {

  prefix "nczerotouch";

  //import ietf-system {
  //  prefix ncsystem;
  //}

  import ietf-netconf-server {
    prefix ncs-server;
  }

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

  contact
  "WG Web:  <http://tools.ietf.org/wg/netconf/>"
  "WG List: <mailto:netconf@ietf.org>"
  "WG Chair: Mehmet Ersue  
             <mailto:mehmet.ersue@nsn.com>"
  "WG Chair: Bert Wijnen  
             <mailto:bertietf@bwijnen.net>"
  "Editor: Kent Watsen  
           <mailto:kwatsen@juniper.net>";

  description
  "This module contains a collection of YANG definitions for configuring NETCONF zerotouch."

  Copyright (c) 2014 IETF Trust and the persons identified as
typedef crypt-hash {
    type string {
        pattern
            '$0$.\*'
            + '/0$\{'a-zA-Z0-9-9./'\}{1,8}$\{'a-zA-Z0-9-9./'\}{22}'
            + '/$5$(rounds=\d+)?\{'a-zA-Z0-9-9./'\}{1,16}$\{'a-zA-Z0-9-9./'\}{43}'
            + '/\$6$(rounds=\d+)?\{'a-zA-Z0-9-9./'\}{1,16}$\{'a-zA-Z0-9-9./'\}{86}';
    }
    description
        "The crypt-hash type is used to store passwords using
        a hash function. The algorithms for applying the hash
        function and encoding the result are implemented in
        various UNIX systems as the function crypt(3)."
    }

A value of this type matches one of the forms:

    $0$<clear text password>
    $<id>$<salt>$<password hash>
    $<id>$<parameter>$<salt>$<password hash>

The "$0$" prefix signals that the value is clear text. When
such a value is received by the server, a hash value is
calculated, and the string "$<id>$<salt>$" or
$<id>$<parameter>$<salt>$ is prepended to the result. This
value is stored in the configuration data store.
If a value starting with ‘$<id>$’, where <id> is not ‘0’, is received, the server knows that the value already represents a hashed value, and stores it as is in the data store.

When a server needs to verify a password given by a user, it finds the stored password hash string for that user, extracts the salt, and calculates the hash with the salt and given password as input. If the calculated hash value is the same as the stored value, the password given by the client is accepted.

This type defines the following hash functions:

<table>
<thead>
<tr>
<th>id</th>
<th>hash function</th>
<th>feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MD5</td>
<td>crypt-hash-md5</td>
</tr>
<tr>
<td>5</td>
<td>SHA-256</td>
<td>crypt-hash-sha-256</td>
</tr>
<tr>
<td>6</td>
<td>SHA-512</td>
<td>crypt-hash-sha-512</td>
</tr>
</tbody>
</table>

The server indicates support for the different hash functions by advertising the corresponding feature.

reference
"IEEE Std 1003.1-2008 - crypt() function
RFC 1321: The MD5 Message-Digest Algorithm
FIPS.180-3.2008: Secure Hash Standard";

{container configlet {
    description
    "Top-level container for ZeroTouch configuration objects.";
    
    container system {
        // no way to use top-level "netconf" container?
        container authentication {
            
            list user {
                key name;
                description
                "The list of local users configured on this device.";
            }
            
            leaf name {
                type string;
                description
                "The user name string identifying this entry.";
            }
            
            leaf password {
            }
        }
    }
}
type crypt-hash;
description
  "The password for this entry.";
}
list ssh-key {
  key name;
description
  "A list of public SSH keys for this user.";
  reference
  "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";

  leaf name {
    type string;
description
      "An arbitrary name for the ssh key.";
  }

  leaf algorithm {
    type string;
    mandatory true;
description
      "The public key algorithm name for this ssh key.
      
      Valid values are the values in the IANA Secure Shell (SSH) Protocol Parameters registry, Public Key Algorithm Names";
  }

  leaf key-data {
    type binary;
    mandatory true;
description
      "The binary key data for this ssh key.";
  }

}
}

container netconf-server {
  // no way to use top-level "netconf" container?

  container ssh {
    uses ncserver:ssh-config;
    // no way to disable "listen" container?
  }
}
container tls {
    uses ncserver:tls-config;
    // no way to disable "listen" container?
}
}

6. Security Considerations

It is not possible to substitute a Configlet created for a different device, since devices assert that the Configlet contains their unique identifier (e.g., serial number).

It is possible to substitute a Configlet created for a device with a different Configlet created for the same device. Generally, unless imposed by the Configlet Signers, there is no limit to the number of Configlets that may be generated for a given device. This could be resolved, in part, by placing a timestamp into the Configlet and ensuring devices do not load Configlets older than some amount, but this requires the devices have an accurate clock when validating a Configlet and for Configlet Signers to not sign a Configlet when another Configlet is still active.

Confidentiality of Configlets loaded over a network is only assured when the device uses a secure networking scheme and validates the Configuration Server’s certificate.

Confidentiality is further provided by using the fingerprint of the device’s entity certificate when doing a Configuration Server lookup, as it is not guessable and thus makes it nearly impossible for an adversary to lookup.

This draft allows devices to try alternate means to load a Configlet before trying the network, so long as they assert physical presence. For instance, a removable USB drive or a near-field communication mechanism. Further, this draft does not require Configlets to be signed, if loaded via a mechanism that asserts physical presence. or require those Configlets to have the device’s unique identifier value set. All of these relaxations in Security are deemed acceptable because physical presence should only be accessible to trusted parties.
This draft allows devices to use insecure schemes when doing a Configuration Server lookup. This is deemed acceptable because the Configlet is tamper-proof, due to being signed, only confidentiality is lost.

This draft entails the device having an X.509 certificate that is used by the NMS to authenticate the device. This certificate and every certificate in the chain leading to the well known trust anchor, should have an expiration date greater than the device’s useful life expectancy. Given the long-lived nature of these device certificates, it is paramount to use a strong key length (e.g., 512-bit ECC). Configlet Signers should deploy Online Certificate State Protocol (OCSP) responders or CRL Distribution Points (CDP) to revoke certificates in case necessary.

This draft mentions using the device’s serial number as its unique identifier in its entity certificate. This is because serial numbers are ubiquitous and prominently contained in invoices and on labels affixed to devices and their packaging. That said, serial numbers many times encode revealing information, such as the device’s model number, manufacture date, and/or sequence number. Knowledge of this information may provide an adversary with details needed to launch an attack. To address this concern, the certificate could contain the hash of the serial number instead, which the NMS could also compute, but doing so is much less intuitive and raises questions if it is just security through obscurity.

It is paramount the device manufacturer ensures the integrity of the device’s list of trust anchors. It should not be possible for anyone other than the manufacturer be able to modify the list of trust anchors. One way to achieve this to sign the list of trust anchors with a private key known only to the manufacturer, and for the matching public key to be stored on tamper-resistant read-only media.

7. IANA Considerations

None

8. Acknowledgements

The authors would like to thank Russ Mundy and Wes Hardaker for brainstorming the solution presented in this draft with us during the IETF 87 meeting in Berlin.

9. References

9.1. Normative References


9.2.  Informative References


Appendix A.  Examples

A.1.  Signed Configlet

This example illustrates a Configlet configuring both a local user account and reverse-SSH. This Configlet includes both the Configlet Signer’s certificate as well as an Intermediate certificate. Note that ‘\’ characters have been added for formatting reasons.

```xml
<?xml version="1.0"?>
<configlet xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-zerotouch">
  <!-- from ietf-system.yang -->
  <system>
```
<authentication>
  <user>
    <name>admin</name>
    <ssh-key>
      <name>admin's rsa ssh host-key</name>
      <algorithm>ssh-rsa</algorithm>
      <key-data>AAAAB3NzaC1yc2EAAAADQABAQABwAgECAgE...gakWVOZ2zgQ8929uWjCW1Glqn2mPibp2Go1</key-data>
    </ssh-key>
  </user>
</authentication>

<!-- from ietf-netconf-server.yang -->
<netconf-server>
  <ssh>
    <call-home>
      <applications>
        <application>
          <name>config-mgr</name>
          <description>
            This entry requests the device to periodically connect to the Configuration Manager application
          </description>
          <servers>
            <server>
              <address>config-mgr1.example.com</address>
            </server>
            <server>
              <address>config-mgr2.example.com</address>
            </server>
          </servers>
        </application>
      </applications>
      <connection-type>
        <periodic>
          <timeout-mins>5</timeout-mins>
          <linger-secs>10</linger-secs>
        </periodic>
        <reconnect-strategy>
          <start-with>last-connected</start-with>
          <interval-secs>10</interval-secs>
          <count-max>3</count-max>
        </reconnect-strategy>
      </connection-type>
    </call-home>
  </ssh>
</netconf-server>
<host-key>
  <name>ssh_host_key_cert</name>
</host-key>
<host-key>
  <name>ssh_host_key_cert2</name>
</host-key>
</host-keys>
</application>
</applications>
</call-home>
</ssh>
</netconf-server>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod
      Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <Reference>
      <Transforms>
        <Transform
          Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
      </Transforms>
      <DigestMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>2xlFdlVifb1snGBLJueZYrLjSUQ==</DigestValue>
    </Reference>
    <SignatureValue>
      HUx3S7TZxGjUGuhazWGRSB9CBMZ0T+tTrBlFOntCki9wU4UOnSw5KMWDvOVwc6dM
      U10J1uJ1gWhSkn+VvWSWZ6q7LT1YyWyNCiXygMVmX4oRd1pT+nqCDxriB4i14VW
      mVhEw1oe83KjT7W/ODJUE7&F9HRpPy9EgxpQX/7wdKSK+4f2uYKSpq2UumW2DIU
      LeK9vNVRQBBbmcF3zZWnnwKH5V4weQ1mW8497AeSYWgSImSetADIONvxvF2jx
      JqzFEalY1zn81BOzVYy14s1RzbN7YmxhN1R3q52WvWvJr2SylrZ/5BpiHYoDeKoD
      HMQMf3H2L06hm5s8rGg==</SignatureValue>
  </SignedInfo>
</Signature>
<X509Data>MIIFKjCCBBKgAwIBAgIBAJANBgkqhkiG9w0BAQsFADAwMRMwEQYDVQQKEApUUE1f
VmVuZG9yMRkFwFyVQQDFBBdW5pcGxVyiXhYHFhYXONNBMB4XDEzMTAyMDExMjIx
MFOxMTExMTAyMDE2MjIxMFowKzE1MBEGAIUEChQVDVBX12bRMVcjeUIMIBGA1UE
AQQLY2hpF8wMDAwMDQwMQAIA4IBDwAwgqEKAoIBAQOf
4hyWgQsf801sZyYQOBj0P8cCHmINPWNs9Fp3QCCB1Pz1YhcDOgyVmqHzrJyZt+7Q
ZTjPse8B0x4d00kr80uc4NWmze40Psz2HAW6GQ2mo=+uUyXUqQFbi3Ekqrdzdkg
Ri6vuadMKacJH8ug9R+cwb/LlpxHlYy2A5fuH4JLPY91lwABtK8eGhF9cGxBYR+
KqZycoV6aaIvd/4NOICNsaGeAjXXxXXoRF5E6HVKsoIYHPFdi+40BmrCuuWy6
1ybCIP5uZz7Oza4j0n/fPb6SEqEa01lzUEW1FQMZYSbClNY57zWNGQ5dFJO20g
Appendix B.  Change Log

B.1.  00 to 01

Complete re-write.  Switched from using signed DNS records using DNSSEC to using signed YANG-defined XML files using XML Signature. This update took into a lot a feedback from both operators and vendors.

Appendix C.  Open Issues

C.1.  How to best structure the Configlet YANG module?

The current YANG module must redefine parts of the "ietf-system" and "ietf-netconf-server" modules. Also, when referencing parts that it can, the YANG module unnecessarily includes parts it doesn’t need, such as configuring the device to listen for inbound connections. Ideally "deviation" statements could be used to delete the unwanted sub-trees.

C.2.  Should Configlets always be signed?

This drafts states that Configlets don’t have to be signed when loaded through a mechanism that asserts physical presence. However, some have voiced concern, saying that no possible backdoor should be allowed.

Authors’ Addresses

Kent Watsen
Juniper Networks
EMail: kwatsen@juniper.net

Stephen Hanna
Juniper Networks
EMail: shanna@juniper.net

Joe Marcus Clarke
Cisco Systems
EMail: jclarke@cisco.com
Mikael Abrahamsson
T-Systems

EMail: "mikael.abrahamsson@t-systems.se