Network Working Group V. Cridlig
Internet-Draft Universite Henri Poincare Nancy 1
Intended status: Standards Track November 15, 2006
Expires: May 19, 2007

NETCONF access control framework
draft-cridlig-netconf-rbac-00.txt

Status of this Memo

Copyright Notice

Abstract

This document defines a role-based access control framework for the NETCONF configuration protocol.

Table of Contents

1. Requirements notation

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

2. Introduction

2.1. General overview

The current document defines an access control framework for NETCONF. This framework relies on the Role-Based Access Control (RBAC) model and intends to protect NETCONF manager access to resources. The access control process intentionally removes the underlying protocol options from the decisions. In particular, it does not depend on whether the underlying protocol is BEEP, SSH or SOAP. It assumes that this protocol is secure by especially providing authentication, integrity and confidentiality. This choice was made because it is not reasonable to have one access control policy for each of this protocol and for each of their internal options or security services. Another design choice is that each NETCONF agent stores its own access control policy like in VACM. Consequently, the RBAC policy remains customizable for each NETCONF agent in order to fulfill all requirements and scenarii.

A NETCONF agent has one instance of the RBAC policy that applies to all the supported datastores (running, candidate, startup).

2.2. Role-Based Access Control in NETCONF

The RBAC model has many advantages over the other access control models. In particular, it is scalable in terms of maintainability even with a huge number of users. RBAC helps to organize the permissions into higher level job functions called roles. As an example in the NETCONF context, a role can be a firewall management role, a security management role, a routing management role. Roles can be specialized in a hierarchy. For example, the security management role can be a senior role of the firewall management role and therefore, inherits the permissions of this junior role (firewall management role). In the context of this document, an RBAC user is a NETCONF manager. The latter can be assigned to one or several roles. RBAC is fairly well adapted to Netconf because it includes the concept of session, within which permissions can be activated dynamically. An RBAC session is mapped to a NETCONF session.

The NETCONF agent stores an RBAC policy, containing the user, role and permission definitions and their assignements.

Role-Based Access Control model

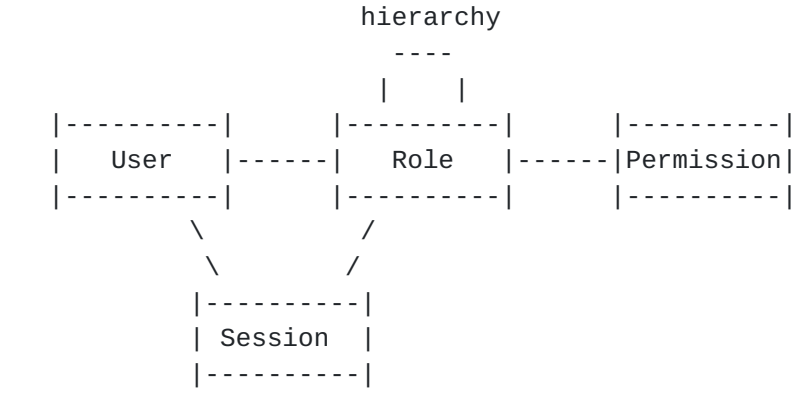

Figure 1

This RBAC policy MAY be stored in an XML file and MAY be part of the agent data model. Therefore the RBAC policy MAY be manageable through NETCONF itself, as it is for VACM in SNMPv3. This allows a large-scale deployement of the access control policy. This document defines a XML-based data model to store the RBAC policy.

The specified model and the involved new operations defined in the current document are covered by a new capability: "urn:ietf:params:xml:ns:netconf:rbac:1.0", abbreviated as "#rbac" in the rest of the document.

3. RBAC-related operations

In the dynamic RBAC model, roles must be activated at run time. Indeed, the permissions of an RBAC user are not constantly available. An RBAC user has to activate a role to be granted the related permissions. This decreases misconfiguration risks since only a subset of the configuration data is accessible at a given time. However, this subset can be the whole configuration datastore if the role allows it. Two operations are necessary to allow role activation and deactivation. These operations are sent by the manager to the agent and are wrapped into an <rbac> operation. The <rbac> operation is a direct child of the NETCONF <rpc> operation.

3.1. Activating a role

<activate-role>

Description:

Activating a role allows a user to be granted the permissions related to this role in the current NETCONF session. This operation can happen at any time after the hello exchange is done. The manager sends an activate-role operation containing a specified role name to the agent. The agent checks its local RBAC policy to decide whether the user can activate this role or not. In order to activate a role, a user MUST be assigned to this role and the role must be enabled. An enabled role is a role that can be activated.

Parameters:

Role name:

The name of the role to be activated.

Positive Response:

If the user is assigned to the requested role, and the role is not activated yet in the current session, then the agent sends an <rpc-reply> containing an <ok> element.

Negative Response:

An <rpc-error> element is included within the <rpc-reply> if the request cannot be completed for any reason. A role activation will fail if the user is not assigned to the requested role, or if the role is already activated or if any other constraint prevents the user from activating the role.

3.2. Deactivating a role

<deactivate-role>

Description:

Deactivating a role allows a user to revoke the permissions related to this role in the current NETCONF session. This operation can happen at any time after the hello exchange is done. The manager sends a deactivate-role operation containing a specified role name to the agent. The agent checks its local RBAC policy to decide whether the user can deactivate this role or not.

Parameters:

Role name:

The name of the role to deactivate.

Positive Response:

If the role was activated in the current session, then the agent sends an <rpc-reply> containing an <ok> element.

Negative Response:

An <rpc-error> element is included within the <rpc-reply> if the request cannot be completed for any reason. A role deactivation will fail if the user had not activate the role previously in the session.

3.3. Role (de)activation typical use case

Role (de)activation process:

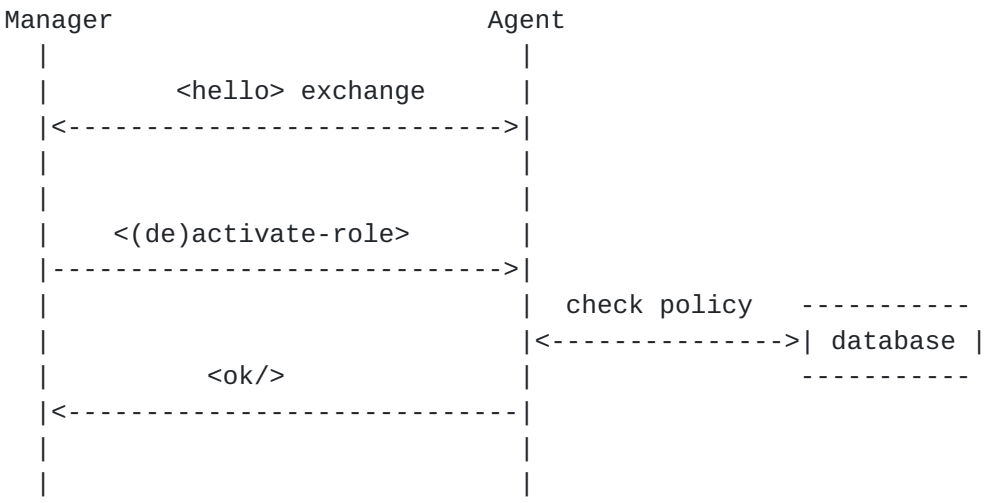

Figure 2

When a NETCONF session is terminated (for example, after a <close-session> or a <kill-session>), then all active roles MUST be deactivated within the current session.

When a NETCONF session is started (after the hello exchange), then a set of default roles MAY be activated by default on a per-user base, without requiring the <activate-role> operation. This default activation on Netconf session startup saves some messages and avoid the need of a user activity. The set of roles activated by default MAY be chosen by the manager.

4. RBAC policy data model

This section defines an XML-based data model representing an RBAC policy. It specifies the main components of RBAC: users, roles and permissions. Sessions are not part of the saved data model, since it is the dynamic part of RBAC. Sessions state (activated and deactivated roles) always remains in memory.

4.1. RBAC components definition

The following items list the components of the RBAC model and their internal relationships (assignments):

User: A user consists of the following attributes:

login: the name of the user.

password: the password of the user.

public-key: the public key of the user.

Role: A role consists of the following attributes:

name: the name of the current role.

junior-roles: a set of junior roles of the current role.

Permission: A permission consists of the following parameters:

scope: a representation of the protected resource, typically a MIB name (could be an XPath expression for more granularity). <scope> is a child of <permission>

operation: read (r), write (w) or notify (n).

Permission-to-Role Assignement (pra):

permRef: reference to a permission.

roleRef: reference to a role.

User-to-Role Assignement (ura):

userRef: reference to a user.

roleRef: reference to a role.

4.2. RBAC policy example

This example consists of 5 roles. RoutingManager role has read access to the routing configuration. InteriorRoutingManager and ExteriorRoutingManager roles inherit the privileges of RoutingManager. InteriorRoutingManager can also write on OSPF and RIP configurations. ExteriorRoutingManager can write on BGP configuration. SuperRoutingManager inherits all previous roles and can write on all routing configurations. SuperManager has full privileges on all the configuration.

RBAC policy example

```
<?xml version="1.0" encoding="UTF-8"?>
<rbac xmlns="urn:loria:madynes:ensuite:yencap:module:RBAC:1.0">
  <prefixes>
    <prefix name="ycp"
            value="urn:loria:madynes:ensuite:yencap:1.0"/>
    <prefix name="bgp"
            value="urn:loria:madynes:ensuite:yencap:module:BGP:1.0"/>
    <prefix name="rip"
            value="urn:loria:madynes:ensuite:yencap:module:RIP:1.0"/>
    <prefix name="ospf"
            value="urn:loria:madynes:ensuite:yencap:module:OSPF:1.0"/>
    <prefix name="ac"
            value="urn:loria:madynes:ensuite:yencap:module:RBAC:1.0"/>
  </prefixes>

  <users>
    <user id="4">
      <login>netconf</login>
      <password>netconf</password>
      <public-key keytype="rsa">AAAAB3NzaC1yc2E...
              ...50RfDJ6M=</public-key>
    </user>
  </users>

  <roles>
    <role id="1">
      <name>RoutingManager</name>
    </role>
    <role id="2">
      <name>InteriorRoutingManager</name>
      <junior-roles>
        <junior-role roleRef="1"/>
      </junior-roles>
    </role>
    <role id="3">
```

```
        <name>ExteriorRoutingManager</name>
        <junior-roles>
          <junior-role roleRef="1"/>
        </junior-roles>
      </role>
      <role id="5">
        <name>SuperRoutingManager</name>
        <junior-roles>
          <junior-role roleRef="2"/>
          <junior-role roleRef="3"/>
          <junior-role roleRef="4"/>
        </junior-roles>
      </role>
      <role id="6">
        <name>SuperManager</name>
        <junior-roles>
          <junior-role roleRef="5"/>
        </junior-roles>
      </role>
    </roles>

    <permissions>
      <permission id="1" op="r">
        <scope>/ycp:netconf/ycp:routing</scope>
      </permission>
      <permission id="2" op="w">
        <scope>/ycp:netconf/ycp:routing/bgp:bgp</scope>
      </permission>
      <permission id="3" op="w">
        <scope>/ycp:netconf/ycp:routing/rip:rip</scope>
      </permission>
      <permission id="4" op="w">
        <scope>/ycp:netconf/ycp:routing/ospf:ospf</scope>
      </permission>
      <permission id="5" op="w">
        <scope>/ycp:netconf/ycp:routing</scope>
      </permission>
      <permission id="6" op="rw">
        <scope>/ycp:netconf</scope>
      </permission>
      <permission id="7" op="w">
        <scope>/ycp:netconf/ycp:routing/ospf:ospf</scope>
      </permission>
    </permissions>

    <uras>
      <ura roleRef="5" userRef="4"/>
      <ura roleRef="6" userRef="4"/>
```

```
    </uras>

    <pras>
      <pra roleRef="1" permRef="1"/>
      <pra roleRef="2" permRef="3"/>
      <pra roleRef="2" permRef="4"/>
      <pra roleRef="3" permRef="2"/>
      <pra roleRef="5" permRef="5"/>
      <pra roleRef="6" permRef="6"/>
    </pras>
  </rbac>
```

Figure 3

4.3. XML schema for RBAC

Here is the XML schema for the RBAC policy.

RBAC XML Schema

```
<xs:schema
  targetNamespace="urn:madynes:params:xml:ns:netconf:rbac:1.0"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="urn:madynes:params:xml:ns:netconf:rbac:1.0">

  <xs:element name="rbac" type="rbacType"/>

  <xs:complexType name="prefixesType">
    <xs:sequence>
      <xs:element name="prefix" type="prefixType" minOccurs="0"
                  maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="prefixType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="name" type="xs:string" use="required"/>
        <xs:attribute name="value" type="xs:string" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="userType">
    <xs:sequence>
```

```
      <xs:element name="login" type="xs:string"/>
      <xs:element name="password" type="xs:string"/>
      <xs:element name="public-key" type="public-keyType"
                    minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:integer"/>
  </xs:complexType>

  <xs:complexType name="public-keyType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="keytype" type="keytypeType" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:simpleType name="keytypeType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="rsa"/>
      <xs:enumeration value="dsa"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="roleType">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="junior-roles" type="junior-rolesType"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:integer"/>
  </xs:complexType>

  <xs:complexType name="usersType">
    <xs:sequence>
      <xs:element name="user" type="userType" minOccurs="0"
                    maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="rolesType">
    <xs:sequence>
      <xs:element name="role" type="roleType" minOccurs="0"
                    maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="permissionsType">
    <xs:sequence>
      <xs:element name="permission" type="permissionType" minOccurs="0"
```

```
                    maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="permissionType">
    <xs:sequence>
      <xs:element name="scope" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="op">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="r"/>
          <xs:enumeration value="w"/>
          <xs:enumeration value="rw"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="id" type="xs:integer"/>
  </xs:complexType>

  <xs:complexType name="urasType">
    <xs:sequence>
      <xs:element name="ura" type="uraType" minOccurs="0"
                    maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="uraType">
    <xs:attribute name="userRef" type="xs:string"/>
    <xs:attribute name="roleRef" type="xs:string"/>
  </xs:complexType>

  <xs:complexType name="prasType">
    <xs:sequence>
      <xs:element name="pra" type="praType" minOccurs="0"
                    maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="praType">
    <xs:attribute name="permissionRef" type="xs:string"/>
    <xs:attribute name="roleRef" type="xs:string"/>
  </xs:complexType>

  <xs:complexType name="rbacType">
    <xs:sequence>
      <xs:element name="prefixes" type="prefixesType"/>
      <xs:element name="users" type="usersType"/>
```

```
      <xs:element name="roles" type="rolesType"/>
      <xs:element name="permissions" type="permissionsType"/>
      <xs:element name="uras" type="urasType"/>
      <xs:element name="pras" type="prasType"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="junior-rolesType">
    <xs:sequence>
      <xs:element name="junior-role" type="junior-roleType"
                  minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="junior-roleType">
    <xs:attribute name="roleRef" type="xs:string"/>
  </xs:complexType>

</xs:schema>
```

Figure 4

4.4. Updating the RBAC policy

Whenever an update is done in the RBAC policy, it MUST be immediately applied to the current NETCONF sessions.

A user can be removed if and only if it does not appear in any user-to-role assignement. A permission can be removed if and only if it does not appear in any permission-to-role assignement. A role can be removed if and only if it does not appear in any permission-to-role and user-to-role assignement.

If a user is removed from the policy, all its current NETCONF sessions must be killed. If a user-to-role assignement is removed from the policy, this role must be deactivated in all sessions belonging to this user where this role is active. If a permission-to-role assignement is removed, the permission MUST be removed from the set of active permissions in all NETCONF sessions having this role or one of its senior role activated. If a permission-to-role assignement is created, the permission must be activated in every session where the related role or one of its senior role is active.

5. Consequences on existing operations

The #rbac capability does not require any change to the formats of existing NETCONF operations. However, an additional processing is required on the agent side when it receives some request. In all cases, the agent MUST consider the active roles within the current session. These active roles relate to permissions expressed as a tuple (operation, scope). From the set of active permissions, the agent is able to build the corresponding sets of authorized scopes for read "r" and write "w" operations. Scopes are expressed as XPath expressions.

5.1. get and get-config operations

When processing a get or get-config operation, some changes are required due to the RBAC policy enforcement. After or before applying the subtree filtering (or XPath) request to the desired configuration, an additional processing MUST take place to apply the RBAC policy. It consists of selecting the allowed XML nodes and pruning the others depending on the currently active roles.

All children and parents of the selected nodes are marked as authorized nodes. While the propagation to the top nodes is necessary to keep the structure of the document, the propagation to the children is a consequence of the subtree-based access control arbitrary choice. It is more scalable to select subtrees rather than individual nodes in the access control policy. Only positive authorizations are allowed in order to avoid conflicts and to constraint the security manager to explicitely give access to resources, all remaining resource being unauthorized by default. All nodes marked as authorized are kept in the XML reply. All other nodes are pruned.

5.2. edit-config operation

As for the get/get-config, when a NETCONF manager is authorized to write on an XML node, she is also allowed to modify the whole subtree whose root is this XML node. On receipt of an edit-config request, the agent applies the XPath expressions of the write "w" permissions set on the request. All children and parents of the selected nodes are marked as authorized nodes. If a "replace", "create", "delete", or "merge" operation is set in one of the parents of the selected nodes, access is denied. This requires to set the operation attribute as close as possible to the node to update.

5.3. copy-config operations

To perform a copy-config operation, a NETCONF manager MUST have write "w" access on the root node of the configuration datastore.

5.4. delete-config operations

To perform a delete-config operation, a NETCONF manager MUST have write "w" access on the root node of the configuration datastore.

6. (SSH, BEEP, SOAP/SSL)-user to RBAC-user mapping

The SSH user name is directly mapped to the RBAC user name, whatever is the authentication method (password or public/private key).

7. Security Considerations

None.

8. Normative References

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

Author's Address

Vincent Cridlig
Universite Henri Poincare Nancy 1

Email: cridligv@loria.fr

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).