Abstract. The NETCONF remote network configuration protocol currently lacks an access control model. The need for such a model has been recognised within the NETCONF working group. The eXtended Access Control Markup Language (XACML) is an XML-based access control standard, with widespread acceptance from the industry and good open-source support. This document proposes a profile that defines how to use XACML to provide fine-grain access control for NETCONF commands.

Keywords: Access control, Configuration management, XACML, NETCONF
1 Introduction

The NETCONF protocol definition [1] specifies in its Security Considerations (section 9) that "This document does not specify an authorization scheme, ... Implementors SHOULD provide a comprehensive authorization scheme with NETCONF". In this document a profile is defined and explained that allows to use the eXtended Access Control Markup Language [2] as authorization scheme for NETCONF commands. The reasons why the use of XACML is suggested are the following:

- XACML is an open standard that has been developed by an industry consortium.
- XACML is an XML [3] based approach, that is well adapted to the authorization challenges encountered within NETCONF.
- XACML is widely accepted and used in a number of commercial products [4].
- Open-source implementations of the XACML standard are readily available.

2 XACML overview

This section gives a short overview of what XACML is and how it works. We only describe the parts of XACML that are needed in this draft, therefore some descriptions may not reflect the full functionality of the corresponding XACML element. Some familiarity with the terms from RFC 2904 [5] (e.g. PDP, PEP) is expected from the reader. The references also include a more detailed introduction [6].

The XACML standard defines two things:

- A XML schema defining a syntax for requests, access control policies and responses.
- A processing model that specifies how a request shall be evaluated by a PDP against a set of policies in order to generate a response.

A request is a collection of attributes typically describing the requesting subject, the requested resource and the action that the subject wishes to perform on the resource. An attribute can for example be a role of the user or a resource group-id.

A policy consists of a target and one or more rules generating an effect. The target describes for which request the policy applies, in terms of conditions on a set of attributes. During evaluation these attributes are
fetched from the request and from external information sources (PIPs) available to the PDP. If a policy applies, its effect will either be PERMIT or DENY.

An example policy target is shown here:

```xml
01 <Target>
  02 <AnyOf>
    03 <AllOf>
      04 <Match MatchId="string-equal;">  
       05 <AttributeValue
         DataType="string">print</AttributeValue>
       06 <AttributeDesignator Category="Action;"  
         AttributeId="resource-id;"  
         DataType="string"/>
       07 </Match>
      10 <Match ... > ... </Match>
    12 <AllOf>...</AllOf>
  14 <AnyOf> ... </AnyOf>
15 </Target>
```

The Target consists of one or more AnyOf sections which must all be fulfilled in order for the policy to apply. Within the AnyOf section, one of the AllOfs must apply. Within an AllOf, all of the Matches must apply. A Match specifies a matching function (which must have a boolean result), and the function parameters, which can either be static AttributeValue (as in line 04) or a value fetched externally from the request or a PIP (as in line 05-07). The AttributeDesignator (lines 05-07) specifies which external value(s) to fetch by giving a Category, an Identifier and a DataType. If several values are returned, only one needs to satisfy the Match function.

### 3 NETCONF overview

The NETCONF configuration protocol describes a set of operations that read or write configuration data on a network device. These operations are transferred to the device by the means of remote procedure calls (RPCs) encoded in XML. The different protocol operations are:

- `<get-config>` allows to get specific parts of a specific configuration.
- `<edit-config>` allows to edit specific parts of a specific configuration.
– `<copy-config>` allows to overwrite a specific configuration with a new one from a specific source.
– `<delete-config>` allows to delete a specific configuration.
– `<lock>` allows to lock a specific configuration for editing.
– `<unlock>` allows to unlock a specific configuration.
– `<get>` allow to get specific parts from the "running" configuration.
– `<close-session>` allows to close your own session.
– `<kill-session>` allows to kill someone else’s session.

For a more specific description of the NETCONF protocol please refer to the NETCONF standard [1].

The present document defines an access control model for these operations and for the extensions defined in the standard.

4 Policy and Request profile for XACML

The goal of this section is to define how a PEP SHOULD generate a XACML request from a RPC carrying a NETCONF operation. The response to this request determines whether the RPC is processed or discarded. Furthermore this profile defines how policies corresponding to permissions about a specific NETCONF operation on specific data SHOULD be formulated. A strong familiarity with the latest XACML syntax is required to fully appreciate this section.

The part of XACML authorisation that deals with the subjects (e.g. in terms of user groups or roles) is out of scope for this profile, since it is not in any way specific to the NETCONF protocol. Thus all the following definitions omit the subject parts of both requests and policies (indicated by "..."). This part can be defined independently from this profile.

4.1 Abbreviations and namespaces

Since XML in general, and especially the XACML syntax, are quite verbose we have defined a set of abbreviations, that can be found in section A. Furthermore we use the term 'Attribute' exclusively for XACML attributes. If we refer to attributes of XML elements we specify this by adding the prefix 'XML' as in 'XML-attribute'.

In order to clearly identify new XACML functions, attributes, and data-types defined specifically for this profile they SHALL have the identifier-prefix "xacml-netconf". Thus the following identifiers-prefixes SHALL be used:
4.2 New XACML functions, attributes and data-types

This section defines the new functions, attributes and data-types for XACML introduced by this profile.

- XACML function: Id="xacml-netconf:function:xpath-node-match"
  Parameter 1 data-type: &xpath; Parameter 2 data-type: &xpath;
  The basic idea of this function is to check whether a xml-node in a xml-document is matched by a certain XPath [7]. Due to the difficulty of encoding a xml-node in its document context, we use a second XPath to point to that xml-node in the document. This function works in two steps. First it evaluates the second xpath (representing the xml-node) against the Content element of the Request. This XPath must match a single xml-node only (otherwise an error is returned). In the second step, the first XPath expression is evaluated against the same Content. If the resulting set of xml-nodes contains the xml-node that results from the first step or one of its ancestors, the value of this function is true, otherwise it is false. The following example shall illustrate how this function works:

Given the function parameters:
<xpath>/top/interfaces[name="Ethernet"]/interface</xpath> and
<xpath>/top/interfaces[name="Ethernet"]/interface</xpath>
as well as the content XML document:

```xml
01 <top>
02  <interfaces>
03   <name>Ethernet</name>
04   <interface>
05     <name>Ethernet0/0</name>
06     <mtu>1500</mtu>
07   </interface>
08   <interface>
09     <name>Ethernet1/1</name>
10     <mtu>3000</mtu>
11   </interface>
12   <interface>
13     <name>Ethernet2/2</name>
```
We get the following evaluation:
1. The second xpath points out the xml-node at line 09.
2. The first xpath points out the xml-nodes 03, 07 and 11. Since xml-node 07 is an ancestor of xml-node 09 the function evaluates to true.

- **XACML attribute: Id="xacml-netconf:attribute:rpc-target"**
  Category=&Resource; data-type=&string;
  This attribute indicates which data store the operation is targeting. In the case of the copy-config command, this is the destination data store. For the lock/unlock commands this is the target data store. An example value would be "running".

- **XACML attribute: Id="xacml-netconf:attribute:rpc-source"**
  Category=&Resource; data-type=&string; or data-type=&AnyURI;
  This attribute indicates the source for the operation. In the case of the copy-config command, this is the source data store or an URL. An example value would be "candidate".

- **XACML data-type: Id="xacml-netconf:data-type:xpath-expression"**
  This data-type is a XPath. The data-type also encodes necessary namespace information, if the XPath is to be used on a namespace aware document. The encoding is an XML-tag with the name of "xpath" containing zero or more attributes, each defining a namespace prefix to namespace URI matching for use with this XPath. An example xpath expression would look like this:

```
<AttributeValue DataType="&xpath;">
  <xpath xmlns:ns1="http://example.com/schema/1.2/config">
    //ns1:top/ns1:interfaces[ns1:name="Ethernet"]
  </xpath>
</AttributeValue>
```
4.3 get and get-config RPC

The get/get-config RPCs get a special treatment, because it was deemed that the whole RPC shouldn’t fail just because the user is not authorised to read parts of the result. Instead the desired behaviour in such a case is to prune the results that are not covered by the users rights. Therefore it is RECOMMENDED to perform access control on the result of a get/get-config RPC instead of on the RPC itself, so that the unauthorised elements can be filtered out and only the authorised ones remain.

Requests for get/get-config RPCs SHALL be formed as follows: As a first step, calculate which xml-nodes of the data model are the results of the RPC. For each xml-node in the result, run an XACML request. The request contains

1. The whole result document under the <Content> xml-node.
2. A ”&Resource;” category Attribute with AttributeId=&resource-id; having the AttributeValue of DataType=”&xpath;” which contains an XPath that uniquely identifies the xml-node in question.
3. Another &Resource; category Attribute with AttributeId=”xacml-netconf:attribute:rpc-source” and the AttributeValue with DataType=”&string;” that identifies the data-model this RPC uses as source (i.e. ”running”, ”startup” or ”candidate”) corresponding to the RPC source. This SHALL always have a value of ”running” for ”get” RPCs.
4. An ”Action” category Attribute with AttributeId=”action-id” and the AttributeValue ”read” with DataType=”&string;”.

Each xml-node that is permitted by the corresponding request is included in the result together with its ancestors and descendants. Furthermore the requests for its descendants are skipped. Those xml-nodes that are not permitted are not included.

These multiple XACML requests can be executed very efficiently if the PDP is running locally on the network element, preferably in the same process as the NETCONF agent. Such an architecture would mean that no new XML documents get generated and no network communication needs to be done for those repeated requests.

For security reasons it is advisable not to report that parts of the response where pruned by access control, otherwise an attacker could use get/get-config to gather information about the existence of parts of the configuration that is not accessible according to the attackers rights.
It is RECOMMENDED that a policy designed to apply to a get or get-config RPC SHOULD match one AttributeValue corresponding to the desired subtree of the data-model with the DataType="&xpath;", the AttributeId="&resource-id;" and the Category="&Resource;". The same <AllOf> element SHOULD contain a match of an AttributeValue corresponding to the desired RPC source (i.e. "running", "startup" or "candidate") with the DataType="&string;", the AttributeId="xacml-netconf:attribute:rpc-source" and the Category="&Resource;". If the policy is to apply to "get" RPCs only this value SHOULD be "running" Furthermore the policy SHOULD match the AttributeValue "read" with the DataType="&string;", the AttributeId="action-id" and the Category="Action" in a <AllOf> element enclosed by a separate <AnyOf> element of the policy target.

Example request:

```
 01 <Request>
 02  ...
 03  <Attributes Category="&Resource;">  
 04   <Attribute AttributeId="&resource-id;">  
 05   <AttributeValue DataType="&xpath;"  
 06     <xpath xmlns:ns1="http://example.com/schema/config">  
 07       /ns1:top[1]/ns1:interface[1]  
 08     </xpath>  
 09   </AttributeValue>  
10  </Attribute>  
11  <Attribute AttributeId="xacml-netconf:attribute:rpc-source">  
12    <AttributeValue DataType="&string;">running</AttributeValue>  
13  </Attribute>  
14  </Attributes>  
15  <Attributes Category="Action">  
16   <Attribute AttributeId="action-id">  
17     <AttributeValue DataType="&string;">read</AttributeValue>  
18   </Attribute>  
19  </Attributes>  
20  <Content>  
21     <top xmlns="http://example.com/schema/config">  
22       <interface>
Example policy:

```xml
01 <Policy PolicyId="ex1"
     RuleCombiningAlgId="&permit-overrides;">
02  <Target>
03     ...
04     <AnyOf>
05         <AllOf>
06             <Match
                           MatchId="xacml-netconf:function:xpath-node-match">
07                 <AttributeValue DataType="&xpath;">
08                   <xpath
                           xmlns:ns1="http://example.com/schema/config">
09                      /ns1:top/ns1:interface
10                    </xpath>
11                 </AttributeValue>
12                 <AttributeDesignator Category="&Resource;"
                               AttributeId="&resource-id;"
                               DataType="&xpath;"/>
13             </Match>
14             <Match MatchId="&string-equal;">
15                 <AttributeValue
                               DataType="&string;">running</AttributeValue>
16                 <AttributeDesignator Category="&Resource;"
                               AttributeId="xacml-netconf:attribute:rpc-source"
                               DataType="&string;"/>
17             </Match>
18         </AllOf>
19     </AnyOf>
20 </Policy>
```
4.4 edit-config RPC

Requests for edit-config RPCs SHALL be formed as follows: Under the 
<Attributes Category="&Resource;" /> element an Attribute with the 
AttributeId="&resource-id;" and the DataType="&xpath;". The 
AttributeValue SHALL be "/[@operation and not(ancestor::*[@ operation])]/".

The same Category SHALL also include an Attribute with the At-
tributeId="&scope;" and the DataType="&string;". The AttributeV-
alue SHALL be "XPath-expression".

Still under the same Category there SHALL be an Attribute with the 
AttributeId="xacml-netconf:attribute:rpc-target" and the Data-
Type="&string;". The AttributeValue SHALL be either "running", "startup" or 
"candidate" corresponding to the RPC target.

Furthermore the request SHALL include the <Attributes 
Category="Action" /> element, containing a single Attribute with the 
AttributeId="action-id" having the Data-
Type="&string;" and the At-
tributeValue of "write".

From the RPC, the contents of the <config> element SHALL be in-
cluded in the request under the <Content> element. If the RPC con-
tains a <default-operation> element the contents of the RPC <config> 
element that are added to the request <Content> element SHALL be 
edited to add a xml-attribute "operation" with a value corresponding to 
the value of the <default-operation> element.

This request format makes use of the Multiple resource profile of 
XACML [8] where the multiple resources are the elements of the RPC 
that have an "operation" xml-attribute and no ancestor with such an
xml-attribute. Using this profile, no access control is performed for operations that have an ancestor operation. This is due to the fact that all edit-config operations are subsumed under the action "write" as far as access control is concerned. The underlying assumption of this profile is that if you are authorised to write to a xml-node in the data-model you are automatically authorised to write to all its children too.

The XPath "/@operation and not(ancestor::*[@operation])" performs the selection of operations with no ancestor operation. If any edit-config operation of the RPC is not permitted, the whole RPC SHALL be denied.

If the RPC uses the :url capability (i.e. a <url> element appears instead of the <config> element), the NETCONF agent SHALL preprocess the RPC by downloading the file pointed to by the URL and replacing the <url> element by a <config> element containing the content of the file.

Another special case that needs to be treated is the following: According to the protocol specification, it is possible to create a syntactically correct edit-config RPC with no operation at all (i.e. specifying 'none' as <default-operation> and having no 'operation' xml-attributes in the <content>). Such an RPC SHALL be discarded according to this profile and not be processed by the NETCONF agent, to avoid leaking information with the error messages.

It is RECOMMENDED that a policy designed to apply to an edit-config RPC SHOULD match one AttributeValue corresponding to the desired subtree of the data-model with the DataType="&xpath;", the AttributeId="&resource-id;" and the Category="&Resource;". The same <AllOf> element SHOULD contain a match of an AttributeValue corresponding to the desired RPC target (i.e. "running", "startup" or "candidate") with the DataType="&string;", the AttributeId="xacml-netconf:attribute:rpc-target" and the Category="&Resource;".

Furthermore the policy SHOULD match the AttributeValue "write" with the DataType="&string;", the AttributeId="&action-id;" and the Category="&Action;" in a <AnyOf> element enclosed by a separate <AnyOf> element of the policy target.

Example request:

01 <Request>
02 ...
03   <Attributes Category="&Resource;">
04     <Attribute AttributeId="&resource-id;">

Example policy:

```
01 <Policy
PolicyId="ex2" RuleCombiningAlgId="&permit-overrides;">
02   <Target>
03     ...
04   <AnyOf>
05     <All0f>
```
06     <Match
07         MatchId="xacml-netconf:function:xpath-node-match">
08         <AttributeValue DataType="&xpath;">
09             <xpath
10                 xmlns:ns1="http://example.com/schema/config">
11                 /ns1:top/ns1:interface[ns1:name="Ethernet0/0"]
12             </xpath>
13     </AttributeValue>
14     <AttributeDesignator Category="&Resource;"
15         AttributeId="&resource-id;"
16         DataType="&xpath;"/>
17     </Match>
18     <Match MatchId="&string-equal;">
19         <AttributeValue
20             DataType="&string;">running</AttributeValue>
21         <AttributeDesignator Category="&Resource;"
22             AttributeId="xacml-netconf:attribute:rpc-target"
23             DataType="&string;"/>
24     </Match>
25     </AnyOf>
26     <AnyOf>
27         <AllOf>
28             <Match MatchId="&string-equal;">
29                 <AttributeValue
30                     DataType="&string;">write</AttributeValue>
31                 <AttributeDesignator Category="Action"
32                     AttributeId="action-id"
33                     DataType="&string;"/>
34         </Match>
35         </AllOf>
36     </AnyOf>
37     </Target>
38     <Rule RuleId="PermitRule" Effect="Permit"/>
39 </Policy>

4.5 copy-config and delete-config RPC

Requests for copy-config and delete-config RPCs SHALL be formed as follows: The RPC target parameter SHALL be included under the <Attributes Category="&Resource;" > element as a single Attribute with
the AttributeId="xacml-netconf:attribute:rpc-target". If the RPC target is one of \{running, startup, candidate\} the DataType SHALL be "&string;" otherwise it SHALL be "&AnyURI;". The AttributeValue SHALL be equal to the RPC target.

In case of copy-config RPCs the request SHALL also include the RPC source under same Category as a single Attribute with the AttributeId="xacml-netconf:attribute:rpc-source". If the RPC source is one of \{running, startup, candidate\} the DataType SHALL be "&string;" otherwise it SHALL be "&AnyURI;". The AttributeValue SHALL be equal to the RPC source.

Furthermore the request SHALL include the 
<Attributes Category="Action"> element, containing a single Attribute with the AttributeId="action-id" having the DataType="&string;". This SHALL have the AttributeValue of "write" if the RPC target is one of \{running, startup, candidate\}. If the RPC target is a URL then this AttributeValue SHALL be "read". When the target is a URL, no configuration data is overwritten, such RPC must therefore be considered 'read' operations. However when the target is a local configuration, the RPC must be considered a 'write' operation.

It is RECOMMENDED that a policy designed to apply to a copy-config/ delete-config RPC SHOULD match one or more AttributeValues corresponding to the desired RPC targets with AttributeId="xacml-netconf:attribute:rpc-target" and the Category="&Resource;". For those targets that are "running", "startup" or "candidate" the DataType SHALL be equal to "&string;", for URL targets, the DataType SHALL be equal to &AnyURI. Each desired RPC target SHOULD be placed in a separate <AllOf> element under a single common <AnyOf> element in the policy target.

The policy SHOULD also match one or more AttributeValues corresponding to the desired RPC sources with the DataType="&string;" or DataType="&AnyURI;", the AttributeId="xacml-netconf:attribute:rpc-source" and the Category="&Resource;". Each desired RPC source SHOULD be placed in a separate <AllOf> element under a single common <AnyOf> element in the policy target. Furthermore if any target elements where one of \{running, startup, candidate\}, then the policy SHOULD match the AttributeValue "write" with the DataType="&string;", the AttributeId="action-id" and the Category="Action" in a separate <AllOf> element enclosed by a separate <AnyOf> element in the policy target.
If any source elements where one of \{running, startup, candidate\},
then the policy SHOULD match the AttributeValue "read" with the
DataType="&string;", the AttributeId="action-id" and the
Category="Action" in a separate <AllOf> element. This element should
be enclosed the same <AnyOf> element as previous "write" action matches
if any, otherwise it is to be enclosed by a separate <AnyOf> element in
the policy target.

Example request:

01 <Request>
02 ...
03 <Attributes Category="&Resource;">
04    <Attribute
05      AttributeId="xacml-netconf:attribute:rpc-target">
06      <AttributeValue
07         DataType="&string;">running</AttributeValue>
08    </Attribute>
09    <Attribute
10      AttributeId="xacml-netconf:attribute:rpc-source">
11      <AttributeValue
12         DataType="&AnyURI;">https://user@example.com:
13         passphrase/cfg/new.txt</AttributeValue>
14    </Attribute>
15  </Attributes>
16 <Attributes Category="Action">
17    <Attribute AttributeId="action-id">
18      <AttributeValue
19         DataType="&string;">write</AttributeValue>
20    </Attribute>
21 </Attributes>
22 </Request>

Example policy:

01 <Policy PolicyId="ex3"
    RuleCombiningAlgId="&permit-overrides;">
02 <Target>
03 ...
04 <AnyOf>
05  <AllOf>
06     <Match MatchId="&string-equal;"
4.6 lock and unlock RPC

Requests for lock/unlock RPCs SHALL be formed as follows: The RPC operation target SHALL be included under the `<Attributes Category="&Resource;">` element as a single Attribute with the AttributeId="xacml-netconf:attribute:rpc-target" and the DataType="&string;". The At-
tributeValue SHALL be either "running", "startup" or "candidate" corresponding to the RPC operation target.

Furthermore the request SHALL include the <Attributes Category="&Action;"> element, containing a single Attribute with the AttributeId="&action-id;" having the DataType="&string;" and the AttributeValue of either "lock" or "unlock" depending on the type of RPC.

It is RECOMMENDED that a policy designed to apply to a lock/unlock RPC SHOULD match one or more AttributeValues corresponding to the desired RPC targets (i.e. "running", "startup" and/or "candidate") with the DataType="&string;", the AttributeId="xacml-netconf:attribute:rpc-target" and the Category="&Resource;". Each desired RPC target SHOULD be placed in a separate <AllOf> element under a single <AnyOf> element in the policy target.

Furthermore the policy SHOULD match both AttributeValues "lock" and "unlock" with the DataType="&string;", the AttributeId="&action-id;" and the Category="&Action;", in separate <AllOf> elements enclosed by a separate <AnyOf> element of the policy target.

Example request:

```
01 <Request>
  02 ... 
  03 <Attributes Category="&Resource;"> 
  04 <Attribute
          AttributeId="xacml-netconf:attribute:rpc-target">
  05 <AttributeValue
          DataType="&string;">running</AttributeValue>
  06 </Attribute>
  07 </Attributes>
  08 <Attributes Category="&Action;">
  09 <Attribute AttributeId="&action-id;">
  10 <AttributeValue
          DataType="&string;">lock</AttributeValue>
  11 </Attribute>
  12 </Attributes>
  13 </Request>
```

Example policy:

```
01 <Policy PolicyId="ex4"
          RuleCombiningAlgId="&permit-overrides;"> 
  02 <Target>
```
4.7 kill-session RPC

Requests and policies for this RPC are defined to be independent of the session-id. Although it would be easily possible to make session-id specific policies and requests, no reasonable use-case for such a feature was found.
Any kill-session RPC SHALL be translated to a request that includes the `<Attributes Category="Action">` element, containing a single Attribute with the AttributeId="action-id" having the DataType="&string;" and the AttributeValue "kill-session". It is RECOMMENDED that a policy designed to apply to a kill-session RPC SHOULD match the single AttributeValue "kill-session" with the DataType="&string;", the AttributeId="action-id" and the Category="Action" in a `<AnyOf>` element of its Target.

Example request:

```xml
01 <Request>
02 ...  
03 <Attributes Category="Action">  
04   <Attribute AttributeId="action-id">  
05     <AttributeValue  
06       DataType="&string;">kill-session</AttributeValue>  
07   </Attribute>  
08 </Attributes>  
09 </Request>
```

Example policy:

```xml
01 <Policy PolicyId="ex5"  
   RuleCombiningAlgId="&permit-overrides;">  
02  
03 ...  
04 <AnyOf>  
05   <AllOf>  
06     <Match MatchId="&string-equal;">  
07       <AttributeValue  
08         DataType="&string;">kill-session</AttributeValue>  
09     </Attribute>  
10     <AttributeDesignator Category="Action"  
11       AttributeId="action-id"  
12       DataType="&string;"/>  
13   </Match>  
14 </AllOf>  
15 </AnyOf>  
16 </Target>  
17 <Rule RuleId="PermitRule" Effect="Permit"/>  
18 </Policy>
```
4.8 close-session RPC

For this RPC it was deemed that no XACML profile was necessary. This results from the assumption that only the person that opened a session should be allowed to submit this RPC to the NETCONF agent. It seems reasonable to expect that the NETCONF agent can enforce this behaviour without the support of the access control system.

4.9 commit and discard-changes RPC

If the NETCONF agent supports the :candidate capability, Any commit or discard-changes RPC SHALL be translated to a request that includes the <Attributes Category="Action"> element, containing a single Attribute with the AttributeId="action-id" having the DataType="&string;" and the AttributeValue of either "commit" or "discard-changes".

It is RECOMMENDED that a policy designed to apply to a commit or discard-changes RPC SHOULD match the single AttributeValue "commit" or "discard-changes" with the DataType="&string;", the AttributeId="action-id" and the Category="Action" in a <AllOf> element enclosed by a <AnyOf> element of its Target.

Example request:

01 <Request>
02 ...
03 <Attributes Category="Action">
04  <Attribute AttributeId="action-id">
05   <AttributeValue
06     DataType="&string;">commit</AttributeValue>
07  </Attribute>
08 </Attributes>
09 </Request>

Example policy:

01 <Policy PolicyId="ex6"
   RuleCombiningAlgId="&permit-overrides;">}
02 <Target>
03 ...
04 <AnyOf>
05  <AllOf>
06   <Match MatchId="&string-equal;">
07     <AttributeValue
4.10 validate RPC

If the NETCONF agent supports the :validate capability, requests for lock/unlock RPCs SHALL be formed as follows: The RPC operation target SHALL be included under the `<Attributes Category="&Resource;"` element as a single Attribute with the `AttributeId="xacml-netconf:attribute:rpc-target"` and the `DataType="&string;"`. The AttributeValue SHALL be either "running", "startup" or "candidate" corresponding to the RPC operation target.

Furthermore the request SHALL include the `<Attributes Category="Action">` element, containing a single Attribute with the `AttributeId="action-id"` having the `DataType="&string;"` and the `AttributeValue` of "validate".

It is RECOMMENDED that a policy designed to apply to a validate RPC SHOULD match one or more AttributeValues corresponding to the desired RPC targets (i.e. "running", "startup" and/or "candidate") with the `DataType="&string;"`, the `AttributeId="xacml-netconf:attribute:rpc-target"` and the `Category="&Resource;"`. Each desired RPC target SHOULD be placed in a separate `<AllOf>` element under a single `<AnyOf>` element in the policy target.

Furthermore the policy SHOULD match the AttributeValue "validate" with the `DataType="&string;"`, the `AttributeId="action-id"` and the `Category="Action"` in a separate `<AllOf>` element enclosed by a separate `<AnyOf>` element of the policy target.

Example request:

```xml
 01 <Request>
 02 ...
 03 <Attributes Category="&Resource;">
 04   <Attribute
```
AttributeName="xacml-netconf:attribute:rpc-target">
05      <AttributeValue
06          DataType="&string;">candidate</AttributeValue>
07      </Attribute>
08  </Attributes>
09  <Attributes Category="Action">
10      <Attribute AttributeId="action-id">
11          <AttributeValue
12              DataType="&string;">validate</AttributeValue>
13      </Attribute>
14  </Attributes>
15  </Request>

Example policy:

01  <Policy PolicyId="ex7"
02      RuleCombiningAlgId="&permit-overrides;">
03      ...
04      <AnyOf>
05          <AllOf>
06              <Match MatchId="&string-equal;">
07                  <AttributeValue
08                      DataType="&string;">candidate</AttributeValue>
09                  <AttributeDesignator Category="&Resource;"
10                      AttributeId="xacml-netconf:attribute:rpc-target"
11                      DataType="&string;"/>
12              </Match>
13          </AllOf>
14      </AnyOf>
15      <AnyOf>
16          <AllOf>
17              <Match MatchId="&string-equal;">
18                  <AttributeValue
19                      DataType="&string;">validate</AttributeValue>
20                  <AttributeDesignator Category="Action"
21                      AttributeId="action-id"
22                      DataType="&string;"/>
23              </Match>
24          </AllOf>
25      </AnyOf>
5 Practical consequences for NETCONF implementations

This profile does not make any assumptions on the data-model that a NETCONF operation affects. However writing a correct policy according to this profile requires such knowledge. This is due to the fact that XPaths matching parts of the data-model have to be inserted in the policy.

A PDP using this profile to perform access control on NETCONF operations will need access to the RPC and for <get> or <get-config> operations, to the results of the RPC. No access to actual device data is required by this profile. If a special treatment for get/get-config proves to be undesirable, a more restrictive interpretation can be implemented by performing a similar access control evaluation as for edit-config RPCs.

This profile makes heavy use of XPath [7] to reference elements in a data-model. It may be the case that XPath processing proves to be too slow for time-critical applications. Therefore alternatives can be considered, such as the Subtree Filtering proposed in the Netconf standard section 6 [1]. This profile can be adapted to such alternatives with relative ease, by creating a new data-type for XACML representing the xml-node selection expression and a new function for XACML equivalent to the "xpath-node-match" Function.

According to this profile, no specific access control architecture is required (i.e. where the PDP and PEP are implemented). However it seems both advisable and possible to have the PDP running at the same location as the NETCONF agent. Although calls to distant PDPs are possible the response time would be prohibitive. In order to allow for fast communication one should aim to have the PDP running in the same process as the NETCONF agent. Our implementation of the XACML PDP is around 300 KByte large and has a memory consumption in the order of magnitude of 700 KBytes (this is mostly due to XML processing at startup). Further optimisation of these numbers is possible if need arises.

If the NETCONF agent supports the :url capability, edit-config RPCs need to be preprocessed to substitute a possible <url> element by a <config> element containing the contents of the file pointed to by the URL.
6 Security Considerations

Depending on the error messages returned by unsuccessful edit-config operations, an attacker might probe parts of the data model that are not covered by the attackers access rights. Especially the ’data-exists’ and ’data-missing’ errors could leak information about the device data. If these leaks are considered severe, one should consider replacing the error message e.g. with an ’operation-failed’ error message without further description.

New capabilities advertised by NETCONF agents can provide new methods of accessing data on the device. If the access control model does not cover such capabilities it is RECOMMENDED to deny requests using them until the model has been extended to cover them.

In order to implement such behaviour for new NETCONF operations, a deny-biased PDP can be used. Such a PDP denies all requests for which no applicable policy can be found.

If the capability affects existing NETCONF operations, the specific profile for these operations SHOULD be extended.

Security considerations from the XACML standard [2] and from the NETCONF standard [1] SHOULD be applied to any use of this profile.

References

A  Abbreviations

For abbreviating XACML policies and requests this profile provides a list of entity declarations, that is to be used within this document. The syntax and expansion for such entities is defined in the XML standard [3] (e.g. &string; will be expanded to “http://www.w3.org/2001/XMLSchema#string”).

- &ENTITY resource-id "urn:oasis:names:tc:xacml:1.0:resource:resource-id”>
- &ENTITY string "http://www.w3.org/2001/XMLSchema#string”>
- &ENTITY string-equal "urn:oasis:names:tc:xacml:1.0:function:string-equal”>
- &ENTITY AnyURI "http://www.w3.org/2001/XMLSchema#anyURI”>
- &ENTITY uri-equal "urn:oasis:names:tc:xacml:1.0:function:uri-equal”>
- &ENTITY xpath "xacml-netconf:data-type:data-type:xpath-expression”>
- &ENTITY scope "urn:oasis:names:tc:xacml:2.0:resource:scope”>

B  Examples

In this section we give examples of requests, policies and their evaluation.

B.1  Get-config example

Given the following get-config RPC:

```xml
01 <rpc message-id="101"
02 xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
03   <get-config>
04     <source><running/></source>
05     <filter type="subtree">
06       <top>
07         <interfaces/>
```
we get the following result before access control:

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <top>
      <interfaces>
        <interface>
          <name>Ethernet0/0</name>
          <mtu>1500</mtu>
        </interface>
        <interface>
          <name>Ehternet1/1</name>
          <mtu>3000</mtu>
        </interface>
        <interface>
          <name>Ethernet2/2</name>
          <mtu>1000</mtu>
        </interface>
      </interfaces>
      <interfaces>
        <interface>
          <name>WLAN0/0</name>
        </interface>
      </interfaces>
    </top>
  </data>
</rpc-reply>
```

Now given and the XACML policy (omitting the subject part):

```
<Policy PolicyId="ex8" RuleCombiningAlgId="&permit-overrides;">
  <Target>
```
...<AnyOf>
  <AllOf>
    <Match MatchId="xacml-netconf:function:xpath-node-match">
      <AttributeValue DataType="&xpath;">path</AttributeValue>
      <AttributeDesignator Category="&Resource;" AttributeId="&resource-id;" DataType="&xpath;"/>
    </Match>
    <Match MatchId="&string-equal;">
      <AttributeValue DataType="&string;">running</AttributeValue>
      <AttributeDesignator Category="&Resource;" AttributeId="xacml-netconf:attribute:rpc-source" DataType="&string;"/>
    </Match>
  </AllOf>
</AnyOf>
</Target>
<Rule RuleId="PermitRule" Effect="Permit"/>
</Policy>

we generate the following requests (omitting the subject part):

...
which is denied.

which is permitted.
which is denied.

which is denied.

which is denied.
which is denied. And finally:

which is also denied. The NETCONF response after access control will therefore like this:
B.2 edit-config example

Given the following edit-config RPC:

```xml
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target><running/></target>
    <config>
      <top>
        <interfaces>
          <name>Ethernet</name>
          <interface operation="replace">
            <name>Ethernet0/0</name>
            <mtu>1500</mtu>
            <ipAddress operation="create">192.0.0.1</ipAddress>
          </interface>
          <interface operation="replace">
            <name>Ethernet1/1</name>
            <mtu>3000</mtu>
          </interface>
          <interface>
            <name>Ethernet2/2</name>
            <mtu operation="replace">1000</mtu>
          </interface>
        </interfaces>
        <interfaces>
          <name>WLAN</name>
          <interface operation="delete">
            <name>WLAN0/0</name>
          </interface>
        </interfaces>
      </top>
    </config>
  </edit-config>
</rpc>
```
30 </config>
31 </edit-config>
32 </rpc>

and the following XACML policy (omitting the subject part):

01 <Policy PolicyId="ex9"
  RuleCombiningAlgId="&permit-overrides;">
02  <Target>
03    ...
04  </AnyOf>
05  </AllOf>
06   <Match
       MatchId="xacml-netconf:function:xpath-node-match">
07     <AttributeValue DataType="&xpath;">
08       <xpath>/top
09          /interfaces[name="Ethernet"]</xpath>
10     </AttributeValue>
11     <AttributeDesignator Category="&Resource;"
12       AttributeId="&resource-id;"
13       DataType="&xpath;="/>
14   </Match>
15   <Match MatchId="&string-equal;">
16     <AttributeValue
17       DataType="&string;">running</AttributeValue>
18     <AttributeDesignator
19       Category="&Resource;"
20       AttributeId="xacml-netconf:attribute:rpc-target"
21       DataType="&string;="/>
22 </Match>
23 </AllOf>
24 </AnyOf>
25   <AllOf>
26     <Match MatchId="&string-equal;">
27       <AttributeValue
28         DataType="&string;">write</AttributeValue>
29       <AttributeDesignator Category="Action"
30         AttributeId="action-id"
31         DataType="&string;="/>
32   </Match>

we generate the following request from the RPC:

```xml
<Request
  xmlns="urn:oasis:names:tc:xacml:3.0:schema:os">
  <Attributes Category="&Resource;">
    <Attribute
      AttributeId="xacml-netconf:attribute:rpc-target">
      <AttributeValue
        DataType="&string;">running</AttributeValue>
    </Attribute>
    <Attribute AttributeId="&scope;">
      <AttributeValue
        DataType="&string;">XPath-expression</AttributeValue>
    </Attribute>
    <Attribute AttributeId="&resource-id;" includeInResult="true">
      <AttributeValue
        DataType="&xpath;">
        <xpath>//*[@operation
          and not(ancestor::*[@operation])]</xpath>
      </AttributeValue>
    </Attribute>
  </Attributes>
  <Attributes Category="Action">
    <Attribute AttributeId="action-id">
      <AttributeValue
        DataType="&string;">write</AttributeValue>
    </Attribute>
  </Attributes>
  <Content>
    <top>
      [Contents of the <top> element in the RPC]
    </top>
  </Content>
</Request>
```

which results in the following XACML response:
<Response xmlns="urn:oasis:names:tc:xacml:3.0:schema:os">
  <Result>
    <Decision>Permit</Decision>
    <Attributes Category="&resource;">
      <Attribute AttributeId="&resource-id;" includeInResult="true">
        <AttributeValue DataType="&xpath;">
          <xpath>/top[1]/interfaces[1]/interface[1]</xpath>
        </AttributeValue>
      </Attribute>
    </Attributes>
    <Status>
      <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
  </Result>

  <Result>
    <Decision>NotApplicable</Decision>
    <Attributes Category="&resource;">
      <Attribute AttributeId="&resource-id;" includeInResult="true">
        <AttributeValue DataType="&xpath;">
          <xpath>/top[1]/interfaces[2]/interface[1]</xpath>
        </AttributeValue>
      </Attribute>
    </Attributes>
    <Status>
      <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
  </Result>

  <Result>
    <Decision>Permit</Decision>
    <Attributes Category="&resource;">
      <Attribute AttributeId="&resource-id;" includeInResult="true">
        <AttributeValue DataType="&xpath;">
          <xpath>/top[1]/interfaces[1]/interface[1]</xpath>
        </AttributeValue>
      </Attribute>
    </Attributes>
    <Status>
      <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
  </Result>
</Response>
<xpath>/top[1]/interfaces[1]/interface[2]</xpath>
</AttributeValue>
</Attribute>
</Attributes>
</Status>
<StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
</Status>
</Result>
<Result>
<Decision>Permit</Decision>
<Attributes Category="&resource;">
<Attribute AttributeId="&resource-id;"
includeInResult="true">
<AttributeValue DataType="&xpath;">
<xpath>/top[1]/interfaces[1]/interface[3]/mtu[1]</xpath>
</AttributeValue>
</Attribute>
</Attributes>
<Status>
<StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
</Status>
</Result>
</Response>

The meaning of this response is the following:

- The operation on line 09 of the RPC is permitted (lines 02-14 of the response).
- There is no applicable policy for the operation on line 25 of the RPC (lines 15-27 of the response). This usually means that we deny the operation and therefore reject the whole RPC.
- The operation on line 14 of the RPC is permitted (line 28-40 of the response).
- The operation on line 20 of the RPC is permitted (line 41-53 of the response).

The operation on line 12 of the RPC is not submitted to access control since a ancestor xml-element already contained a edit-config opera-
tion (line 09). Since we simplify all edit-config operations to 'write’, the operation on line 12 would have been permitted anyway as the ancestor operation was permitted. If the ancestor operation had been denied the whole RPC would be rejected anyway.