HAL
archives-ouvertes.fr

Assessment of security extended XML-based Management

Vincent Cridlig, Humberto Abdelnur, Radu State, Olivier Festor

▶ **To cite this version:**

Vincent Cridlig, Humberto Abdelnur, Radu State, Olivier Festor. Assessment of security extended XML-based Management. [Research Report] 2006, pp.18. <inria-00084004>

HAL Id: inria-00084004
https://hal.inria.fr/inria-00084004

Submitted on 5 Jul 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Assessment of security extended XML-based Management

V. Cridlig H. Abdelnur R. State O. Festor

July 5, 2006

Abstract

The emergence of new management paradigms having XML as a core foundation block demands a comprehensive analysis of their security and performance issues. This paper presents an extension to the existing NetConf protocol. This extension consists of a security architecture and some advanced XML specific features. We describe a series of experiments addressing the performance and operational aspects of our developed implementation and provide grounded answers to issues of significant relevancy to the research community.

1 Introduction

Flexible management paradigms based on XML as an underlying core support emerged over the past years and are currently becoming operational. This fast paced evolution was possible due to the XML data-level interoperability as well as due to the large basis of supporting tools and libraries existing for most programming languages. Although much has been done, some fundamental questions related to the security architecture and more advanced XML specific operations are still open. Our work within this landscape is centered on providing a security framework for XML enabled management frameworks, implementing and validating the IETF endorsed Netconf protocol [10] and validating its appropriateness for complex network management tasks (like for instance the configuration of the BGP routing plane). This is addressed in the first section of our paper.

The second section of this paper introduces some proposed extensions to the Netconf configuration protocol. These extensions are related to a security framework for Netconf, compression and to the edit-config operation for additional XSLT support. The third section of the paper presents the benchmarking results of our implementation and its security extensions. The implementation itself is shortly described in section 4, highlighting its modular architecture. In section 5, an overview of the related works is given. Finally, section 6 concludes the paper.

2 Netconf

2.1 General architecture

Netconf is a XML-based network configuration protocol, proposed by the IETF in order to achieve simplicity and management data hierarchy. Netconf is stream-oriented and relies on a Remote Procedure Call (RPC) mechanism. The client (a Netconf manager) typically issues a request to a server (a Netconf agent) which returns a reply *rpc-reply*. An *rpc* tag always embeds a Network Management operation which can be one of *get*, *get-config*, *edit-config*, *copy-config*, *lock*, *unlock*, *close-session* or *kill-session*.

An important point is that a manager sees the configuration data as a XML document, thus giving him a very high, uniform, easy and clear abstraction level. The operations are conceptually performed on that virtual or real document. No other communication channel is even needed since all services can potentially be configured through Netconf.

Before sending rpc operations, a manager must first establish a Netconf session on the agent. A session is carried over a TCP connection and remains open until a close-session or kill-session is issued to the agent. In order to initialize that session, each party sends a *hello* message containing its capabilities and a session-id is issued by the agent. This message is exchanged asynchronously over the TCP connection. Capabilities are a feature that allows each party to advertise its supported features (data model, vendor-specific operations, etc...). Each capability is identified using a unique URN identifier. Netconf defines a set of capabilities like, for instance, #xpath, #startup, #candidate that can be extended with new ones.

Get and get-config operations allow to retrieve data from the Netconf agents and come with two

possible filtering methods. While the first one, subtree filtering, is specific to Netconf, the second one, XPath, is an independant technology. Figure 1 shows a very simple example that allows to retrieve the software packages installed in the agent. This list of RPM packages is part of the data model. The first request addresses the desired data with an XPath expression. The second one is based on subtree filtering.

Each method has its interest. XPath allows to build fine-grained request like */netconf/system-/rpm-list/rpm[contains(text(),'java')]* which selects all rpm elements that contain *java* into their text value. It is also possible to select the number of output nodes and the text value at different XML level using multiple criteria. Subtree filtering allows to build a kind of template document that is filled by the agent. It is legitime to wonder if one of the methods is more performant than the other, and this question has been actively debated in the IETF working group (see https://psg.com/lists/netconf/). We have performed a series of experiments on this issue and the results are discussed in this paper.

2.2 Extended Netconf

The current IETF specifications [10] represent a pragmatic solutions for XML based network management, making for commonly agreed on standard. We extended the IETF proposed specifications with more advanced features related to security, efficient data transmission and agent site filtering and scoping functionalities. These extensions and their experimental benchmarking are the main focus of this paper.

2.2.1 Security

Extending the Netconf protocol with advanced security features is of crucial importance in a multi-party configuration environment, where large scale infrastructures are managed by multiple administrators having different competencies and security clearance levels. An illustrative example can be considered the management of the BGP [21] routing plane, used to interconnect routing domains called Autonomous Systems (AS), where each such domain can be roughly seen as a standalone administrative domain. Within such a domain, a specific routing policy is implemented to reflect management and business requirements. Routing information is exchanged between these domains using an exterior routing protocol. The most commonly used is the BGP4, supporting basic route reachability advertisement as well as more advanced features related to traffic engineering tasks and policy driven routing. These features are provided by a set of protocol extensions allowing to filter route advertisements, tag specific routes in order to drive the route selection process, and to signal to a peering router local preferences.

These features allow to define a routing policy for an AS. There is a central registry where operators can publish their routing policies and check the ones published by other operators. Unfortunately, as it is well known, this information is very often obsolete and outdated. These routing policies are essentially management data and could be made accessible to peering partners if appropriate access control mechanisms were defined. The real world practice shows that operators tend to implement these policies at the relative low level of abstraction given by router configurations. The de-facto configuration files store in text format the configuration commands. Expressing routing policies with commands (compatible with Command Line Interface, CLI [22]) is typically done with route maps. If appropriate access control were to be defined, an operator should be able to allow a peering partner to access in read only mode, only those policies related to the latter, without leaking additional information related to third party potential competitors. As of today, such a scenario is impossible, and for illustration purposes consider the scenario shown in Figure 2. This network topology consists of three ASs: AS 1, AS 2 and AS 3. Routers R1, R2 and R3 belong to autonomous system AS 1, AS 2 and respectively AS 3. Both R1 and R2 are neighbors of R3.

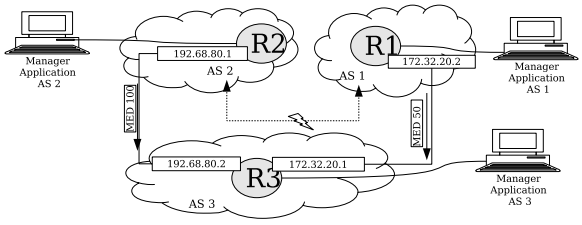

Figure 2: BGP sample topology

A subset of the BGP configuration for router R3 is depicted in Figure 3 in a Command Line Interface (CLI) language. This BGP configuration expresses the following routing policy. First, R3 allows to route BGP packets only if the AS of the source is 3. It means that route advertisement from R1 and R2 will never be forwarded

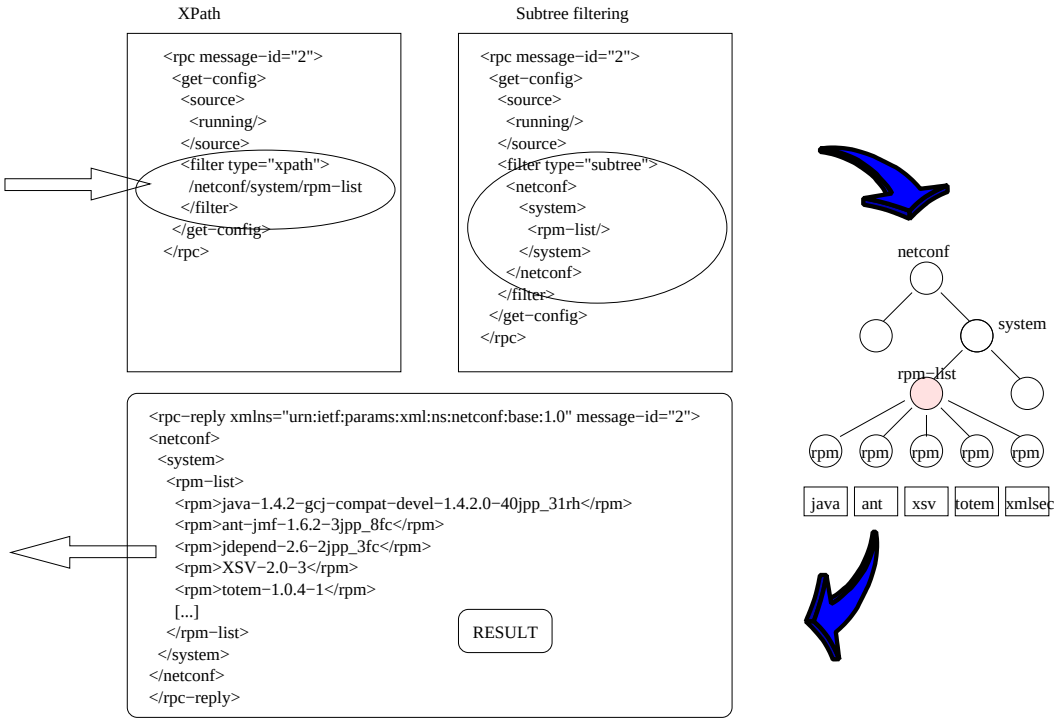

Figure 1: Simple get-config examples for RPM list retrieval

by R3 to other routers. Consequently, R3 will never give any information to R2 about the existence of R1 and vice versa. Second, R3 adds 50 to the weight of the route from R1 to R3. Third, R3 adds 100 to the weight of the route from R2 to R3. Therefore R3 will rather use R1 than R3.

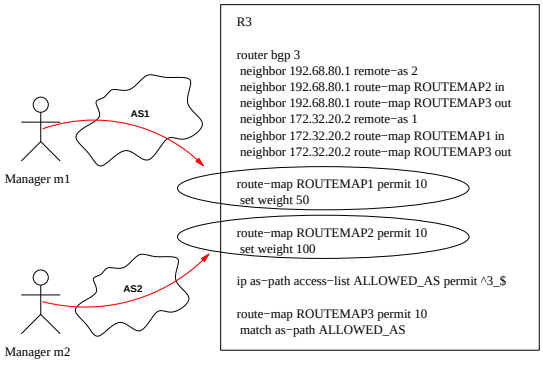

Figure 3: Network Sample CLI configuration

This configuration information is difficult to extend with a flexible access control (at the AS3 site) allowing for instance to allow two managers from AS1 and respectively AS2 to read only the policies related to their own domains.

However, the same configuration settings, can be expressed in an XML data model as shown in Figure 4. Although XML configuration might be more difficult to deal with by an experienced administrator, there are some major advantages from a security point of view. The hierarchical structure of an XML document can be used to implement a flexible access control, where entities could be allowed a per node or per subtree access.

For instance in the previous example, we could allow read only access to the *route-map* nodes.

Integrated Security Framework Figure 5 shows our integrated security components proposed in [7]. The core is split into four main security areas: authentication, integrity, confidentiality and access control. Each area is related to one or more adjacent techniques or concepts. For instance, XML-Encryption is dedicated to confidentiality. Access control means a flexible mechanism to manage access to resources. The flexibility aspect is evaluated regarding the platform dynamicity and size. While dynamicity implies the capability to frequently modify the access rights, the size is dependant on the number of devices and managers. Authentication is the mechanism that guarantees that an entity is the one it claims to be. Integrity is the mechanism that ensures that the management data in transit has not been modified.

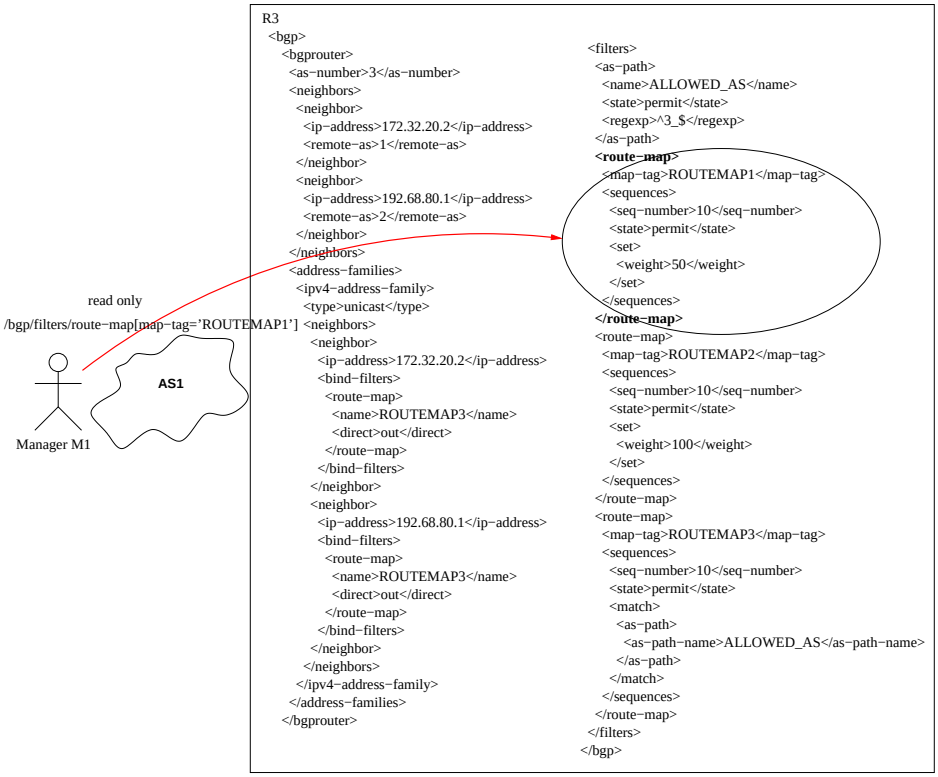

Figure 4: Network Sample XML configuration

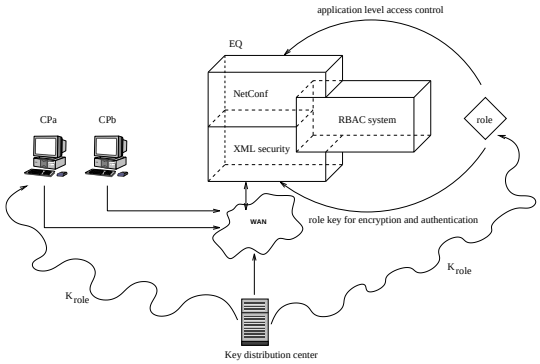

Figure 5: Conceptual security components

In order to address an efficient security plane, we consider a distributed Role Based Access Control (RBAC [14]) type mechanism, in which roles are distributed on the fly. However, access is performed in a non-intermediated way, meaning that the architecture delivers tokens (confidentiality and authentication keys) to principals. To prove the ownership of tokens, an entity signs and cyphers the messages with the keys bound to the roles. In order to manage this dynamicity, we need a fast mechanism for the key exchange system. This is particularly obvious if role revocations are considered.

We consider in this article an XML-based management environment where agents use the Netconf protocol. We will give only a high level presentation of the general architecture. A detailed description can be found in [7]. Therefore, we propose the use of already defined XML security paradigms (XML-Signature [3], XML-Encryption [13]) to perform data access control.

Our security model for Network configuration is built around several main entities depicted on Figure 5. While configuration providers (CP) act like a data configuration source, the managed devices (EQ) run a Netconf agent which receives RPC requests from the different config-

uration providers. For instance, CP_a and CP_b can load different partial configurations onto an agent configuration making for a multi-source configuration. These configurations are loaded with respect to different access rights; an agent must be able to decrypt incoming configurations and to bind them to some access rights. We will describe in this section the mechanisms needed to meet these requirements. An RBAC manager is responsible for security mediation among configuration providers and managed devices. This entity provides the different security services: authentication, data integrity, confidentiality and access control. The RBAC manager hosts RBAC policies which are dynamically deployed on our network entities.

RBAC allows high level and scalable access control process and configuration. The RBAC model consists in a set of users, roles, permissions (operations on resources) and sessions. The originality of the RBAC model is that permissions are not granted to users but to roles, thus allowing an easy reconfiguration when a user changes his activity. A role describes a job function or a position within an organization. The RBAC model allows the description of complex access control policies while reducing errors and administration costs. Introduction of administrative roles and role hierarchies makes it possible to considerably reduce the amount of associations representing permissions to users allocation.

All agents store an RBAC policy describing the permissions that a role is allowed to perform. Figure 16 is an example of such a permission set. In this model, the managers send their requests on behalf of a role. Therefore an agent is able to authenticate and decrypt the requests because it knows the role in use. For each role, the architecture provides one key for authentication and one key for encryption.

After the role authentication and decryption, an agent can start the access control step. According to its local policy and to the role used, it can decide if the request is allowed or not.

In the following, a scenario showing security usage, interests and consequences will be presented in a BGP context. We will consider the topology illustrated in Figure 4. In the scenario, the administrator of AS 3 defined two roles: **roleR1** for the router R1 and **roleR2** for the router R2. In order to allow R1 to read the route-map ROUTEMAP1, a permission must be added to the RBAC policy (see Figure 6). Next, this permission is associated to role roleR1.

There are some other XML-based access con-

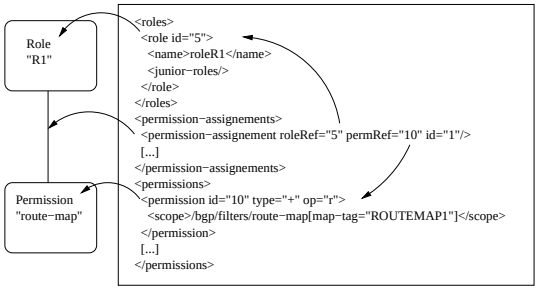

Figure 6: XML-based RBAC policy

trol languages like, for instance, OASIS eXtensible Access Control Markup Language (XACML) [25]. This standard allows to describe access control policies. OASIS also provides a specific profile for RBAC [24]. The reason why we did not write our policy with RBAC profile of XACML is that it has an important restriction compared with the original RBAC model. RBAC profile of XACML does not allow dynamicity. It is not yet possible to activate and deactivate roles within a session. It means that all available roles for a given user are activated by default. Therefore, dynamic separation of duty is not supported.

Our approach allows to grant privileges to some entities and describes the protected resources with XPath language. All requests are authenticated and encrypted with a key. Each key is bound to a role and a key distribution mechanism is leveraged. This architecture is benchmarked later in the paper. A complete review of the security architecture can be found in our paper [7].

Another alternative is SSH ([27]), which however does not include an access control mechanism. SSH is the candidate transport protocol for Netconf, endorsed by the IETF. It is based on PKI and therefore, allows authentication, integrity and encryption. Like XML-Encryption, SSH supports many encryption algorithms like 3DES or AES. A negotiation is used between the client and server to choose the best available algorithm.

2.2.2 Extended filtering and scoping

If we consider the required Netconf request to modify the filters configuration, illustrated in Figure 7.a, such that a final sequence has to be added to every *route-map* whose *map-tag* is different than **ROUTEMAP2**, then the possible solutions offered by the current Netconf specifications are either to make several requests, (one

for each *route-map* see Figure 7.b) or to create a more complex request (as shown in Figure 7.c). Obviously, as more route-map and/or sequences are to be added, the requests become very complex and cumbersome.

Our first extension to the default filtering mechanisms is to extend the syntax of the edit-config to allow a "xpath" attribute in every XML node inside the `<config>` tag. For example, the equivalent request for the previous example is shown in Figure 8 (*Looking into the Values*).

Note that the number of route-maps in the configuration does not affect the size of the request. So rather than explicitly use every value that has to match with the configuration, our approach allows to specify it in a XPath expression.

Our second extension is to design a new operation similar to edit-config accepting a XSLT document rather than a XML as input. In this case the XSLT document will create a valid edit-config by transforming the actual XML configuration document in an standard edit-config XML request. The equivalent configuration of the previous example should look like shown in Figure 8 (Pushing XSLT to generate the edit-config XML).

Note that this proposal has the same advantages than the first extension mechanism. The resulting request document, which will be a standard edit-config, is similar to the one shown in Figure 7 (*XML Possible Request 2*).

A variation of the second extension is to modify the XML configuration by an XSLT request, but in this case, the input XSLT document will transform the current XML configuration to the desired one. An example is illustrated in Figure 8 (*Modify the XML configuration by an XSLT*). Such an approach suits well for configuration deployement task in large scale networks, especially when a master configuration file stored in XML (in an XML proprietary format like JunosScript [18]) has to be adapted to a given network device. From a conceptual point of view, this approach is a limited active networking scheme, where only XSLT expressed code is injected to be executed on an agent.

One issue that is not well specified in the Netconf specification is how an agent should deal with an edit-config request in the search of the nodes which have one ancestor with attribute *operation*. Consider the example illustrated in Figure 9 (*Ambiguity in the request*).

The activated-group and the description may be added to every neighbor or the description may be added to all the neighbors that have set

Ambiguity in the request

```
<bgprouter>
  <neighbors xc:operation="merge">
      <activated-group>
        Group
      </activated-group>
      <description>
        Description Text
      </description>
    </neighbor>
  </neighbors>
</bgprouter>
```

Specifying Search Nodes

```
<bgprouter>
  <neighbors>
    <neighbor xc:operation="merge">
      <activated-group xc:searchnode="true">
          Group
      </activated-group>
      <description>
          Description Text
      </description>
    </neighbor>
  </neighbors>
</bgprouter>
```

Figure 9: Proposal for resolve the Ambiguity Example

an activated-group as **Group**.

Our extension is to resolve this ambiguity by extending the specification of the edit-config and introduce an attribute *searchnode*, in order to make the request explicit. With this extension, the edit-config request for the example shown in Figure 9 (*Ambiguity in the request*) will merge to every neighbor the activated-group and the description. The example of *Specifying Search Nodes* in the same figure adds the description to all the neighbors whose activated-group is equal to group.

2.2.3 Compression extension

As XML language introduces a potentially huge verbosity (see Figure 4), use of compression can dramatically save network bandwidth consumption. Moreover, it gives a simple way to avoid clear text transmissions.

Using compression is of interest, in particular in combination with security features. Since processing time for encryption increases fast with the message size, compression before encryption can reduce encryption processing cost. These issues are illustrated and discussed on the basis of the performance tests presented in the next section.

3 Experimental performance evaluation

To validate and evaluate the proposed security architecture and the compression module, a prototype [8] has been implemented. The benchmarking tests are done locally on a Pentium 4 3.2GHz with 2 Go RAM, see Table 1.

In order to give an overview of the global performances, the agent can manage up to 17 get-config requests per second when no logging mechanism and no security is deployed. When security (encryption, RBAC) and compression are enabled, the performance decreases to 8 re-

XML Configuration Example

```
<bgp>
   ...
   <filters>
      <route-map>
         <map-tag>ROUTEMAP1</map-tag>
         ...
      </route-map>
      <route-map>
         <map-tag>ROUTEMAP2</map-tag>
         ...
      </route-map>
      <route-map>
         <map-tag>ROUTEMAP3</map-tag>
         ...
      </route-map>
   </filters>
</bgp>
```

a)

XML Possible Request 1 (ROUTEMAPX stands for ROUTEMAP1 and ROUTEMAP3 generating in this way 2 requests)

```
<bgp>
   <filters>
      <route-map>
         <map-tag>ROUTEMAPX</map-tag>
         <sequences xc:operation="merge">
            <seq-number>50</seq-number>
            <state>deny</state>
         </sequences>
      </route-map>
   </filters>
</bgp>
```

b)

XML Possible Request 2

```
<bgp>
   <filters xc:operation="merge">
      <route-map>
         <map-tag>ROUTEMAP1</map-tag>
         <sequences>
            <seq-number>50</seq-number>
            <state>deny</state>
         </sequences>
      </route-map>
      <route-map>
         <map-tag>ROUTEMAP3</map-tag>
         <sequences>
            <seq-number>50</seq-number>
            <state>deny</state>
         </sequences>
      </route-map>
   </filters>
</bgp>
```

c)

Figure 7: Motivation XML Example

Looking into the Values

```
<bgp>
   <filters>
      <route-map>
         <map-tag xc:xpath=". != 'ROUTEMAP2'"/>
         <sequences xc:operation="merge">
            <seq-number>50</seq-number>
            <state>deny</state>
         </sequences>
      </route-map>
   </filters>
</bgp>
```

Modify the XML configuration by an XSLT

```
<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="@* | node() ">
      <xsl:copy>
         <xsl:apply-templates
               select="@* | node() " />
      </xsl:copy>
   </xsl:template>

   <xsl:template match="/bgp/filters/route-map/map-tag[. != 'ROUTEMAP2']">
      <map-tag>
         <xsl:value-of select="."/>
      </map-tag>
      <sequences>
         <seq-number>50</seq-number>
         <state>deny</state>
      </sequences>
   </xsl:template>
</xsl:stylesheet>
```

Pushing XSLT to generate the edit-config XML

```
<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="*">
      <xsl:apply-templates select="*" />
   </xsl:template>

   <xsl:template match="/bgp/filters">
      <bgp>
         <filters operation="merge">
            <xsl:for-each select="route-map">
               <xsl:if test="map-tag != 'ROUTEMAP2'">
                  <route-map>
                     <map-tag>
                        <xsl:value-of
                              select="map-tag"/>
                     </map-tag>
                     <sequences>
                        <seq-number>50</seq-number>
                        <state>deny</state>
                     </sequences>
                  </route-map>
               </xsl:if>
            </xsl:for-each>
         </filters>
      </bgp>
   </xsl:template>
</xsl:stylesheet>
```

Figure 8: XML Advanced Configuration Proposal

Description	Caracteristics
RAM memory	2 Gigabytes
CPU	Pentium 4, 3.2 GHz
OS	Linux FC 4, kernel 2.6.12
python	2.4
security libraries	PyXMLSec, paramiko
crypto. keys length	128 bits

Table 1: Test environment caracteristics

quests per second. For the benchmarking, we used only get-config operations. These represent the worst case for security performance because they require a low CPU consumption. If the tests are done on complex requests, the CPU consumption dedicated for security will be negligible compared with the time to perform the Netconf request. With fast requests, it is possible to highlight the influence of security mechanisms on the global processing time. However, we compare compression and encryption performances for both large as well as small sized XML documents.

Figure 10 illustrates different use cases: (1) encryption + compression, (2) encryption, (3) compression, (4) default. The measures are done within the agent. For this test, 7 different get-config requests, applying either subtree filtering or XPath, are performed. Each of them is evaluated 64 times and an average of the processing time is computed. This average is represented on the figure. Each column relates to the total processing time of a get-request.

The results show that compression and decompression time is insignificant compared with the global request processing time. Also RBAC processing takes roughly 6% of the global time. Encryption represents 9.5% and decryption 3.6%. Using both cryptography, compression and access control consumes about 20% of the global time.

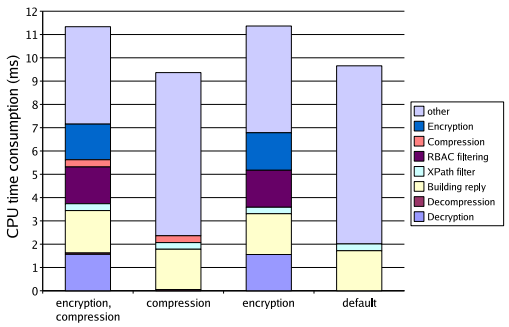

Figure 10: Global request processing time

Note that the total processing time increases by a factor 60 if a XML Schema validator systematically checks the received messages with the Netconf XML schema. The total average processing time is changing from 11 ms to 650 ms. In such a case, the proportion of CPU time dedicated to compression and security decreases. CPU or memory consumption due to XPath technology is of minor importance compared with the use of a XML schema validator. Filtering with XPath takes less then 1 ms. Checking the validity of a Netconf message with XML Schema takes more than 600 ms. These results give an overview of the global performances of this implementation as well as the performance of each functionnality relatively to the others.

Table 2 displays the result response time for different typical data retrieval tests. The agent and the manager are located on two devices connected to a hub and communicate over SSH. For this test only, the agent is located on a laptop centrino 1.6GHz with 1 Gigabytes of memory and runnning Fedora Core 4. The global request processing time per agent remains the same as the previous results, but on the manager side, the results are as follows. The manager is sending 1000 get-config requests using XPath capability and then, the average response time is computed for each. The tested BGP configuration is realistic enough to be used in a real network. While a test on a sample BGP configuration gives a response time of 58.40 ms, a realistic BGP configuration is retrieved in around 200 ms. These results assess the suitability of Netconf for such tasks.

Management data	Response time
network interfaces	38.73
network routes	49.71
BGP configuration	200.08
RBAC policy	43.05

Table 2: Average response time in ms

3.1 On the use of compression

To illustrate the interest of compression, Figure 11 considers different examples for several message sizes. Please note that this scale is not uniform-sized. Due to the very heterogeneous distribution of XML document sizes, a uniform-sized scale would have not fitted in the figure layout. The sizes of many Netconf requests (get-

config using XPath, ok rpc-reply, copy using ftp) range roughly between 100 and 300 bytes. These Netconf messages don't really take advantage of compression. We approximate to at least 50% the pourcentage of such Netconf messages among all processed messages. The size of a hello message with 6 capabilities is 500 bytes. However, the size of a very small RBAC configuration (three users, four roles, 6 permissions and few relationships between them) as well as a extremely basic network interface configuration are about 2500 bytes. Comparatively, the size of a sample BGP configuration is about 5000 bytes but can easily grow to 10000 bytes. The size of the full list of rpms on a typical machine is 25000 bytes. The response size of a get-config on a Netconf agent implementing a larger data model can be evaluated to hundreds of kilobytes. While Figure 11 gives an overview of some typical Netconf message or configuration sizes, some of them can be much greater than the given values. For instance, an edit-config request could reach more than 30 kilobytes.

Figure 12 shows the compression results on arbitrary XML documents. We limited our measurements to the original document size of 25 kilobytes for which the compression rate reaches 65%. These measurements prove the high interest in using compression, since compression significantly decreases the size of transported XML configurations.

Figure 12 also gives the CPU time consumption of compression and decompression, depending on the document size. The average time for compression is 5 times greater than decompression and increases slowly with the document size. In the worst case, which is when the processing time request is very small (simple get-config for instance), compression and decompression processing time take less than 1% of the total message processing.

To conclude on the use of compression, we observed that bandwidth is used efficiently and CPU time consumption is not significant when compared to the global processing time.

3.2 Integrity and confidentiality

Figure 13 gives the sizes of encrypted and decrypted Netconf requests, depending on the original document size. The encryption algorithms are tripledes-cbc and aes128-cbc and both use a 128 bits key. For the tests, we considered a set of sample XML documents having various sizes, and applied 100 times each encryption algorithms for each document. Then, an average is computed for each document. Applying these algorithms on our document set, we observe that the encrypted document size increases linearly with the original document size. While the encrypted document size is the same for both encryption algorithms, aes128-cbc consumes less CPU than tripledes-cbc for both encryption and decryption. Therefore, aes128-cbc, which is recommended by NSA for strength reasons, is the best choice.

Figure 13 also gives the CPU time consumption of the encryption and decryption process, depending on the document size. Encryption and decryption take less than 3 ms for documents sizes less than 10000 bytes. In the worst case, the sum of these times represents 13% of the global Netconf processing time. Again, the worst case occurs when the time to build the XML response is very low. In that case, the CPU consumption proportion of encryption and decryption is maximal.

3.3 Using both compression and encryption

An experiment comparing the processing time of compression, encryption and compression+encryption on XML documents having different sizes led us to the following conclusion: when the size of the document reaches a threshold, it is faster to perform first compression and then encryption than to only encrypt the document. It means that beyond this threshold, using both compression and encryption allows not only to save network bandwidth but also to decrease the agent CPU consumption at the same time.

Figure 14 shows that `compression+encryption` line is crossing the `encryption` line. This event happens when the document size reaches roughly 4 kilobytes, which is actually the size of a very simple BGP configuration.

3.4 Performance of XPath versus subtree filtering

In order to compare the performances of both node selection methods, XPath and subtree filtering, we prepared 8 get-config requests with subtree filtering and their equivalent XPath based requests. The requests are available in *http://www.loria.fr/∽cridligv/-xpathSubtree.html*. We used as many different options as possible in subtree filtering requests: selection nodes, content match nodes, element

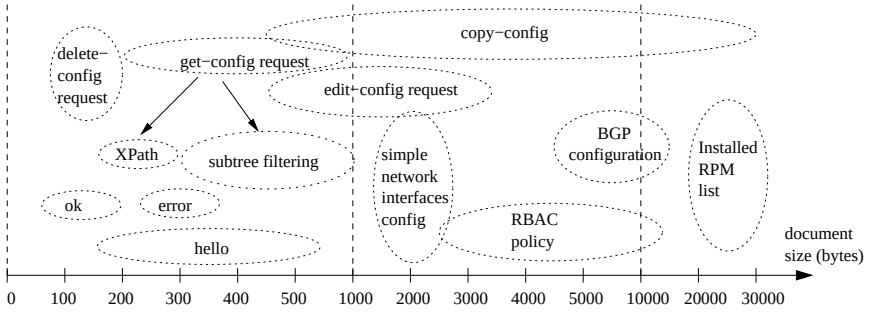

Figure 11: Overview of Netconf message sizes

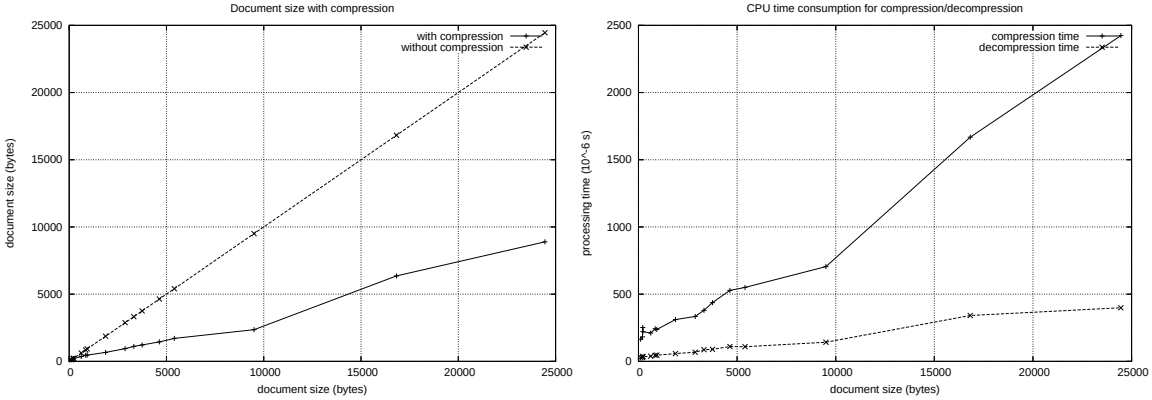

Figure 12: Compression results

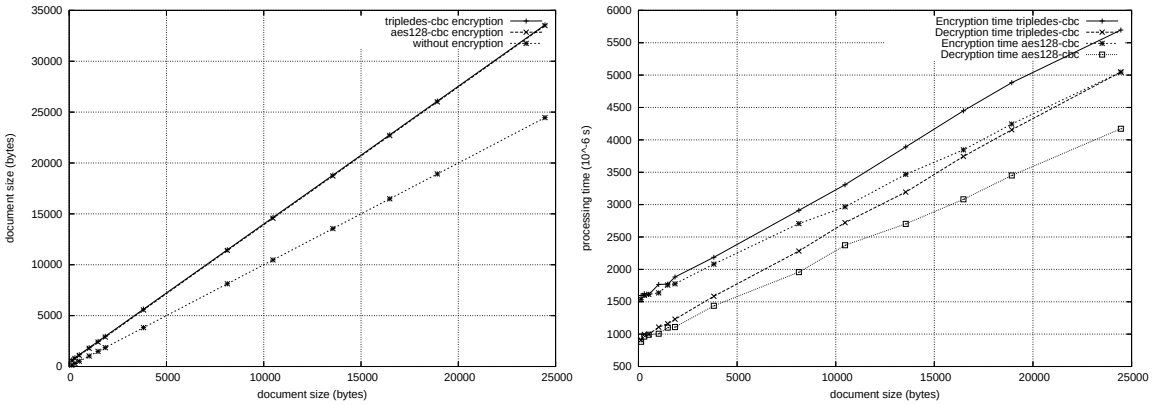

Figure 13: Encryption results

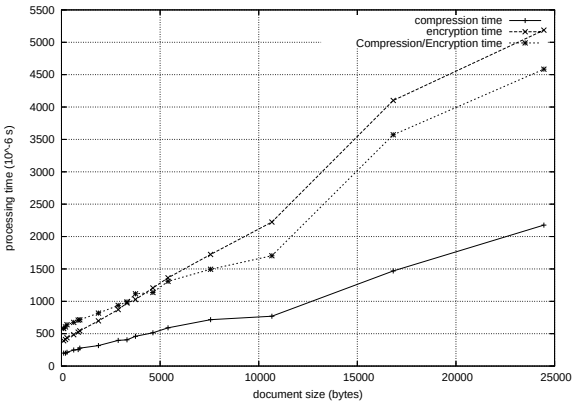

Figure 14: CPU time consumption with different combinations

nodes. Each request is sent 10 times and then an average time is computed. This experiments showed us how subtree filtering is adapted to build a customized document and how XPath is efficient to select nodes on more complex criteria (for instance, conditions on text node value or node number). Subtree filtering is somewhere between XPath and XSLT: it allows to select nodes but also to apply a mask to the configuration. It is always possible to translate a subtree filtering request to XPath but the philosophy is different and both methods are useful and complementary.

The results displayed on Table 3 show that subtree filtering and XPath have similar average processing time. Each request is made 1000 times and an average is computed afterwards. The two filtering approaches are slightly different. When XPath is adapted to select isolated data items, subtree filtering is more suitable to build complex views made of different parts of the config and to visualize them previously in a template-like document. In order to build a document made of completely different parts with Xpath, a manager has to use a | symbol to join the various parts of the selected nodes. However, the subtree filtering can only use absolute addressing while XPath allows both relative and absolute expressions like, for instance, //iface[name='eth0'] or /netconf/network/interfaces/iface[name='eth0']. The advantages of XPath are 1) it consumes less bandwidth since the requests are much more compact than subtree filtering and, 2) it does not require an important effort from the managers.

To weight these results, it is important to note that the total request processing time is much higher than the filtering processing time, whatever be the selection method. To conclude on filtering, both approaches have their advantages and are complementary.

3.5 RBAC model benchmarking

We have also assessed the impact of access control on CPU time consumption. We implemented a RBAC policy with XPath as a resource addressing scheme. In our model, a Netconf request is sent on behalf of a role. The agent is able to build the list of permissions for that role and all its junior roles. The model implements a role hierarchy: a role inherits the permissions of all its junior roles. The results, illustrated on Figure 15, show the additional processing time needed for the access control mechanism. These values also outline the effect of access control mechanism and the number of permissions. The experiments were performed with 0 to 9 permissions. As expected, the CPU time increases linearly with the number of rules.

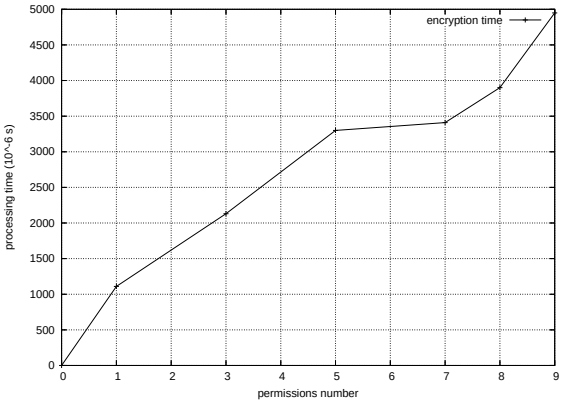

Figure 15: Influence of permissions number

Figure 16 illustrates the set of permissions used for the benchmarking. They are expressed using the XPath syntax. As shown with the permission 1 (allowing read and write operations on all *permissions* with id equal to 5), the access control can also be performed on the access control policy itself.

3.6 Integrated security vs SSH

In order to compare the performances between the two different underlying secure application protocols (our extended XML-based security framework and SSH), a Netconf manager is using both protocols and sending 1000 times the 8th get-config requests using XPath of the Table

Filtering	1	2	3	4	5	6	7	8	9	10	11	12
Subtree	1171	1482	694	463	477	1503	2114	1371	919	775	1733	863
Xpath	1173	1039	1219	194	687	1286	2828	1973	852	916	1332	2368

Table 3: Performances of node selection methods in μs

3. When running over SSH, the total time between the emission of the first request and the reception of the last response is 16.77 seconds. When running over XML-Encryption, it takes 15.62 seconds. Therefore, processing times are quite similar.

SSH session establishment is not taken into account. Similarly, key distribution time for XML-Encryption is not measured here, since this is a stationary phase test.

```
<permissions>
  <permission id="1" type="+" op="rw">
    <scope>/netconf/security/rbac/permissions
    /permission[@id='5']</scope>
  </permission>
  <permission id="2" type="+" op="rw">
    <scope>/netconf/interfaces</scope>
  </permission>
  <permission id="3" type="+" op="rw">
    <scope>/netconf/routing/bgp</scope>
  </permission>
  <permission id="4" type="+" op="rw">
    <scope>/netconf/system</scope>
  </permission>
  <permission id="5" type="+" op="r">
    <scope>/netconf/log</scope>
  </permission>
  <permission id="6" type="+" op="rw">
    <scope>/netconf/security/test/users
    </scope>
  </permission>
  <permission id="7" type="+" op="rw">
    <scope>/netconf/security/test/roles
    /role[@id='2']</scope>
  </permission>
  <permission id="8" type="+" op="rw">
    <scope>/netconf/security/test/roles
    /role[@id='1']/junior-roles</scope>
</permission>
```

Figure 16: Set of permissions

4 Prototyping of a Netconf framework

This section gives an overview of the Netconf test platform. It consists of a command line manager, a web-based manager and an agent. Only the implementation design of the two last ones is presented. In order to allow a more and more completed data model, the architecture can be easily extended with plugins that are called *modules* in this implementation. A detailed overview of the implementation can be found in [6].

4.1 Netconf agent implementation

Figure 17 illustrates the architecture of our Netconf agent. The agent is organized into several functional blocks. The *Socket layer* is in charge of assembling the incoming Netconf messages and also sending prepared messages. The *RPC layer* checks the `<rpc>` level, optionally compress/decompress and encrypt/decrypt the messages. The *Request layer* function is to extract the Netconf operation embedded in the `<rpc>` node. Then it calls the related method (get-config, edit-config, ...) from the *Dispatcher* block.

The *Dispatcher* block is responsible for performing the requested operation. It uses three functional blocks that are called serially. The *Modules Resolver* is able to retrieve the list of modules that can handle the Netconf operation. A *Module Register Table*, storing XPath expressions related to each module, is parsed to bind the data received to the right XPath(). A modules list is returned to the *Dispatcher*. All

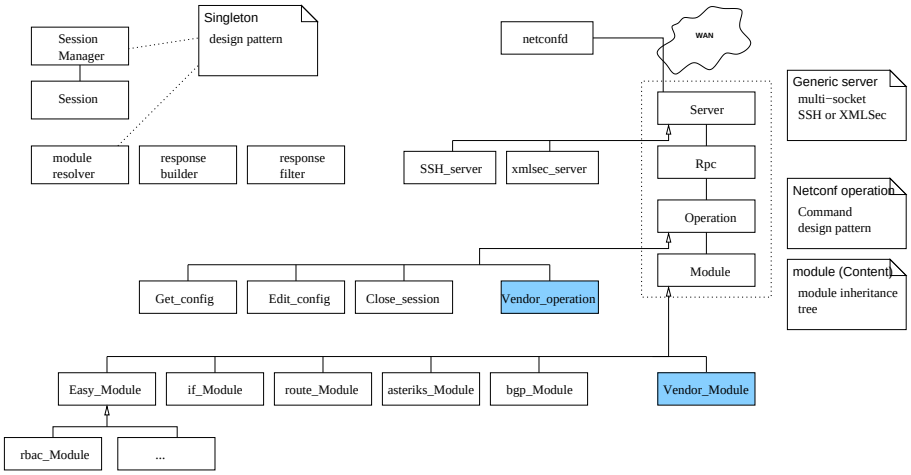

Figure 17: Netconf agent functional architecture

these modules are queried by the *XML Response Builder* to perform the operation. In the case of a *get-config* and if security is activated, the *XML Response Filter* removes all the data that the role is not allowed to read.

In order to extend our agent with a new module, a developer must provide a class that inherits a generic `Module` class and overrides the Netconf operations implemented in `Module`. The `Module` class defines the methods signatures and returns Netconf errors in case the method (for instance get-config) is not implemented in that particular new module. The modules hierarchy is illustrated in Figure 18. The top level class is the generic module implementation. In order to make development of new modules easier, copy-config is generic and is implemented using edit-config. Other methods generate `<rpc-error>` by default, in case the inheriting module does not implement the current method (even if it should). A set of operational modules inherit directly from the `Module` class: one is dedicated to network interfaces management, one enbales BGP router management and one is for firewall rules. `Easy_Module` is a more specific implementation that can be inherited to help Netconf module developers. All modules must be registered in a configuration file in order to be loaded into memory at startup.

Figure 19 illustrates the code of a "route" module. It inherits from `Module` class and implements a getConfig method. EnSuite software, which consists of the manager and the clients, is freely available in our web page: *http://madynes.loria.fr/ensuite*. For a complete presentation, the reader is refered to [6].

5 Related works

In an XML environment, the first network management protocol to propose security mechanisms was Junoscript [18]. It relies on Secure Shell (SSH [27]) to provide authentication and encryption independently of the XML application layer. It also provides mechanisms to manager identity authentication through a login/-password process. The main difference with our approach is that we provide an integrated security management. ACL rules are integrated directly in the XML device configuration document and bound to RBAC roles instead of users for scalability issues. Contrary to SSH, where keys are independent of the application level, the keys for authentication and confidentiality are deeply bound to roles and exchanged dynamically in our architecture. In our approach, a message emitted under a role is encrypted and authenticated with the keys corresponding to that role. This provides a flexible way to integrate authentication, confidentiality and access control in the same architectural plane.

Some intermediate solutions for XML management were proposed with XML/SNMP gateways [26] [20]. These gateways allow XML-based managers to interact transparently with SNMP agents. This provides a quite flexible way to build management application with all the panel of XML tools. However, no security concerns were taken into account apart from using HTTPS between the managers and the gateway. The use of SNMPv3 implies to define user accounts, key distribution and access control policies. In [9], we extend these gateways with security features and provide an end-to-end security

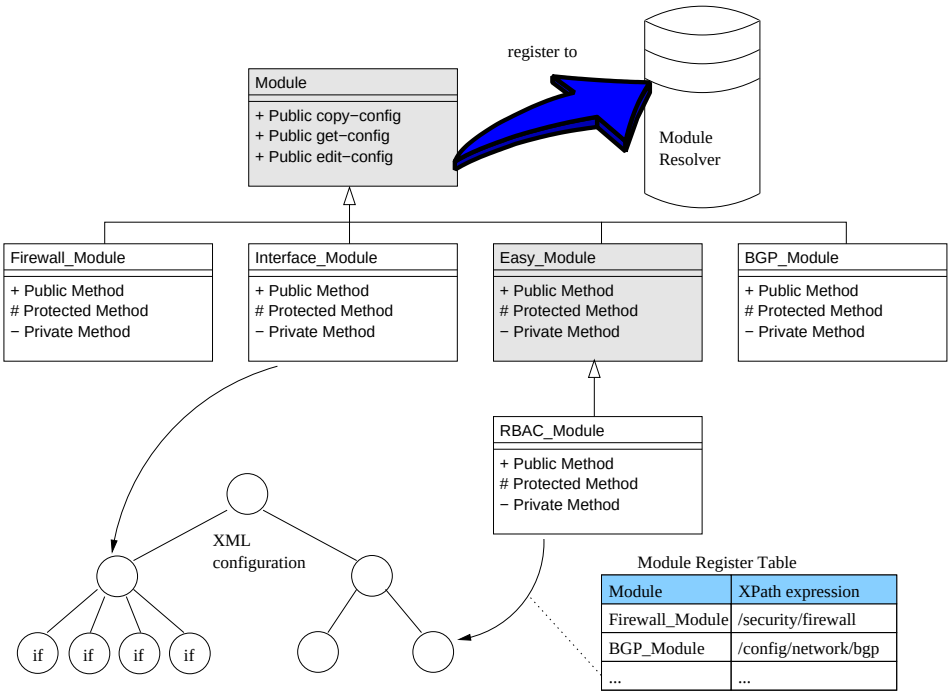

Figure 18: Module inheritance in YencaP

continuum.

We give in this paragraph an overview of the XML security concepts which have been developed mostly for the security needs of web services. Over the past years, some XML languages have been defined to address a set of security requirements. Many existing security mechanisms have been adapted to XML languages so that it is now possible not only to encrypt but also to digitally sign all or parts of an XML document. The World Wide Web consortium published a set of recommendations that describe the syntax of XML tokens and the way of handling such elements. In particular, XML-DigitalSignature [3] describes the syntax for digital signature and its linked information such that the algorithms and keys to use or the way to retrieve them. Moreover, XML-Encryption [13] provides the guidelines to generate encrypted XML elements and to link them with security credentials. XML-Canonicalization aims at describing the preprocessing of an XML element before it is either encrypted or authenticated. Indeed, some equivalent XML elements can be written in different ways depending on the application that handles it. Several parts such as white spaces and namespaces can be changed during the different processes. This introduces some problems when XML elements need to be hashed for authentication. XML-Canonicalization [4] defines a set of rules to format the elements in such a way that it will always produce the same hash. Since we also address distributed XML applications, we use the XML security framework ([3, 13, 4]) to protect Netconf messages.

Our approach differs from a centralized RBAC policy in the sense that the policy is distributed. There is no central RBAC server evaluating all the requests to perform access control on XML documents. We rather use some rules integrated in XML documents. This approach is inspired from SyncML [12] [2]. A similar approach has been submitted recently in the Netconf mailing list [1].

Lupu et al. presented in [17] [15], [16] some policy-based approaches to address the access control and organization issue in the context of distributed systems. The authors described in particular the use of hierarchical roles to organize the principals interacting with a system. They also described the relationships between these roles. Our approach differs in that it uses an existing RBAC model, there is no central authorization server and it is deeply bound to cryptography.

Issues concerning SNMP protocol are presented in [23]. The authors highlight three main problems arising when a manager is retrieving large amounts of MIB data. These problems are *latency*, *network overhead* and *table retrieval*.

```
import os, string
from Ft.Xml.Domlette import NonvalidatingReader, implementation
from Ft.Xml import XPath, EMPTY_NAMESPACE
from Ft.Xml.Domlette import NonvalidatingReader, PrettyPrint
from Modules.modulereply import ModuleReply
from Modules.module import Module

class Route_Module(Module):
    """ Main class of Route module. Allows routing table management. ""

    def __init__(self,parameters):
        """ Create an instance and initialize the structure needed by it."""
        self.dict = ["Destination", "Passerelle", "Genmask", "Indic",
        "Metric", "Ref", "Use", "Iface"]

    def getConfig(self):
        route = self.getRoute()
        self.doc = implementation.createDocument(EMPTY_NAMESPACE, None, None)
        element = self.doc.createElementNS(EMPTY_NAMESPACE,"route")
        self.doc.appendChild(element)

        for routeEntry in route:
            entryelement = self.doc.createElementNS(EMPTY_NAMESPACE,"route-entry")
            element.appendChild(entryelement)
            for data in self.dict:
                datanode = self.doc.createElementNS(EMPTY_NAMESPACE,data)
                textNode = self.doc.createTextNode(routeEntry[data])
                datanode.appendChild(textNode)
                entryelement.appendChild(datanode)

            modulereply = ModuleReply(replynode=self.doc.documentElement)
        return modulereply

    def getRoute(self):
        route = []
        res = os.popen3("route")
        # Read stdout output
        resultat = res[1].read()
        lignes = string.split(resultat, "\n")
        lignes = lignes[2:]

        for ligne in lignes:
            entry = string.split(ligne,' ')
            for i in range(0,entry.count('')):
                entry.remove('')
            if entry != []:
                routeEntry = {}
                for i in range(0, len(self.dict)):
                    routeEntry[self.dict[i]] = entry[i]

            route.append(routeEntry)

        return route
```

Figure 19: Sample module code example

These are mainly due to the way operations such as get-next and get-bulk are defined and also to the data encoding. To deal with these issues, the authors propose some alternative approaches by considering other encodings, compression, protocols and an new get-subtree operation in order to optimize network bandwidth and CPU consumption and easiness to build management application.

In [5], the authors proposed a security performance analysis of SNMPv3. This analysis proves the expected performance degradation when moving from SNMPv2c to SNMPv3. This relative degradation, due to the introduced security features, is to be compared with the huge improvement introduced by security support.

In [19], some performances analysis related to Web Services to SNMP gateways are made. They also studied the influence of using security (HTTP vs HTTPS) and compression (zlib algorithm) on the network bandwidth consumption. The authors propose an interesting study by comparing protocol-level (Get, GetNext) and object-level (GetIfTable) gateways and their influence on bandwidth consumption.

In [11], a comparison between SNMP and CORBA is achieved. The study shows that CORBA is more suitable (bandwidth consumption) for large data retrieval, while SNMP has better performance for a context of small amount of data exchange and limited device memory.

6 Conclusion

This paper proposes a set of extensions for Netconf configuration protocol along with a real life performance evaluation. These extensions address security issues and introduce a new approach for the *edit-config* operation.

The security framework that was proposed in a previous publication [7] is applied for a BGP context. It illustrates a flexible access control policy that grants well-defined and controlled privileges on BGP router configurations. This access control policy assesses who can read or write the configuration data expressed in XML encoding and illustrates the suitability of XPath to address and protect such hierarchical XML resources. The architecture relies on a distributed encryption key system, each key being bound to a role. This means that the access control policy is deeply integrated with the confidentiality and authentication service deployed with XML security standards in a full security framework for Netconf.

The second extension is relative to the use of a existing XML technology to edit XML configuration data. It reviews the edit-config operation and more particularly the merge attribute and points out the advantages of using XSLT instead or complementarily. XSLT, which is specialized in transforming XML documents, is popular in the developpers community and therefore could be enforced more easily by administrators or network management platform developers.

In the second part of the paper, the performance evaluation of the model is fully detailed. The results ensure the feasability of such a security framework, comparing the influence of different options: encryption, compression, access control with our proposed security model and a brief comparison with SSH. The CPU time consumption dedicated to security is acceptable and compression not only dramatically decreases the network overhead introduced by XML encoding but also speeds up the encryption processing time. A comparison with another Netconf implementation would be helpful but no such other open-source implementation exists.

The last part overviews the design and architecture of our free Netconf software suite (EnSuite, http://madynes.loria.fr/ensuite) which consists of a web-based manager and the agent itself. The agent allows plugins addition thanks to a modular architecture and supports SSH as well as our own security model. It also supports network interface, route, BGP configuration management.

References

[1] Sandeep Adwankar. NetConf Data Model. Internet Draft, July 2004.

[2] Open Mobile Alliance. SyncML Consortium. http://www.syncml.org.

[3] Mark Bartel, John Boyer, Barb Fox, Brian LaMacchia, and Ed Simon. XML-Signature Syntax and Processing. W3C Recommendation, February 2002.

[4] John Boyer, Donald E. Eastlake, and Joseph Reagle. Exclusive XML Canonicalization Version 1.0. W3C Recommendation, July 2002.

[5] A. Corrente and L. Tura. Security Performance Analysis of SNMPv3 with Respect to SNMPv2c. In *Proceedings of the 2004*

IEEE/IFIP Network Operations and Management Symposium, NOMS 2004, pages 729–742, 2004.

[6] V. Cridlig, H. Abdelnur, J. Bourdellon, and R. State. A NetConf Network Management Suite: ENSUITE. In *Proceedings of the5th IEEE International Workshop on IP Operations & Management (IPOM 2005)*, October 2005.

[7] V. Cridlig, R. State, and O. Festor. An Integrated Security Framework for XML based Management. In *Proceedings of the Ninth IFIP/IEEE International Symposium on Integrated Network Management (IM 2005)*, IFIP Conference Proceedings, May 2005.

[8] Vincent Cridlig and Radu State. Yencap. http://madynes.loria.fr/ensuite.

[9] Vincent Cridlig, Radu State, and Olivier Festor. Role based access control for XML based management gateway. In *Submitted to the 15th IFIP/IEEE Distributed Systems: Operations and Management, DSOM 2004*, December 2004.

[10] R. Enns. NETCONF Configuration Protocol. Internet Draft, February 2005.

[11] Q. Gu and A. Marshall. Network Management Performance Analysis and Scalability Tests: SNMP vs. CORBA. In *Proceedings of the 2004 IEEE/IFIP Network Operations and Management Symposium, NOMS 2004*, pages 701–714, 2004.

[12] Uwe Hansmann, Riku Mettala, Apratim Purakayastha, and Peter Thompson. *SyncML: Synchronizing and Managing Your Mobile Data*. Prentice Hall PTR; 1st edition, September 2002.

[13] Takeshi Imamura, Blair Dillaway, and Ed Simon. XML Encryption Syntax and Processing. W3C Recommendation, December 2002.

[14] R. Kuhn. Role Based Access Control. NIST Standard Draft, April 2003.

[15] E. Lupu, Z. Milosevic, and M. Sloman. Use of Roles and Policies for Specifying, and Managing a Virtual Enterprise. In *Ninth IEEE International Workshop on Research Issues on Data Engineering: Information Technology for Virtual Enterprises (RIDE-VE'99)*, March 1999.

[16] E. Lupu and M. Sloman. Reconciling Role Based Management and Role Based Access Control. In *Second ACM Workshop on Role Based Access Control (RBAC'97)*, pages 135–142. ACM Press, November 1997.

[17] E. Lupu and M. Sloman. Towards a Role Based Framework for Distributed Systems Management. *Journal of Network and Systems Management*, 5(1):5–30, 1997.

[18] Tony Mauro. *JUNOScript API Guide for JUNOS Release 6.1.* Juniper Networks, 1194 North Mathilda Avenue. Sunnyvale, CA 94089, USA, sonia saruba edition, September 2003.

[19] Ricardo Neisse, Ricardo Lemos Vianna, Lissandro Zambenedetti Granville, Maria Janilce Bosquiroli Almeida, and Liane Margarida Tarouco. Implementation and Bandwidth Consumption Evaluation of SNMP to Web Services Gateways. In *Proceedings of the 2004 IEEE/IFIP Network Operations and Management Symposium, NOMS 2004*, pages 715–728, 2004.

[20] Yoon-Jung Oh, Hong-Taek Ju, Mi-Jung Choi, and James Won-Ki Hong. Interaction Translation Methods for XML/SNMP Gateway. In Metin Feridun, Peter G. Kropf, and Gilbert Babin, editors, *Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2002*, volume 2506 of *Lecture Notes in Computer Science*, pages 54–65. Springer, October 2002.

[21] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). STD 62, http://www.ietf.org/rfc/rfc1771.txt, March 1995.

[22] G. Sackett. *Cisco Router Handbook*. McGraw-Hill Companies; 2nd edition, December 2000.

[23] Ron Sprenkels and Jean-Philippe Martin-Flatin. Bulk Transfers of MIB Data. *The Simple Times*, 7(1):1–7, March 1999.

[24] OASIS Standard. Core and hierarchical role based access control (RBAC) profile of XACML v2.0. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbac-profile1-spec-os.pdf, February 2005.

[25] OASIS Standard. XACML 2.0 Core: eXtensible Access Control Markup Language (XACML) Version 2.0. http://www.oasis-open.org, February 2005.

[26] Frank Strauß and Torsten Klie. Towards XML Oriented Internet Management. In Germán S. Goldszmidt and Jürgen Schönwälder, editors, *Proceedings of the Eighth IFIP/IEEE International Symposium on Integrated Network Management (IM 2003)*, volume 246 of *IFIP Conference Proceedings*, pages 505–518. Kluwer, March 2003.

[27] T. Ylonen and C. Lonvick. SSH Transport Layer Protocol. Internet Draft, June 2004.