AN INTEGRATED SECURITY FRAMEWORK FOR XML BASED MANAGEMENT

V. Cridlig, R. State, O. Festor
LORIA - INRIA Lorraine
615, rue du jardin botanique
54602 Villers-les-Nancy, France
{cridligv, state, festor}@loria.fr

Abstract  This paper proposes an integrated security framework for XML based management in large enterprise networks. Our security framework extends the traditional Role Based Access Control model with a cryptographic mechanism allowing efficient updates of the roles to user associations and its integration within a security framework for XML based management, where access control per managed object, confidentiality, integrity are tightly bound.

Keywords:  security, xml, rbac

1. Introduction

XML-based technologies are increasingly used in the discipline of device network and service management while a lot of effort was successfully invested in protocols and integrated frameworks. Few has been done in the security of this emerging XML-based management plane. This paper addresses exactly this point and provides two contributions.

The main contribution of this paper is to introduce an integrated security framework within a decentralized management approach where several configuration providers can manage multiple devices. We assume that XML is the basis of the management plane, and a configuration at one agent can be made of configuration parts coming from many sources. This requires a minimal organization in particular for security and efficiency reasons. In particular, only well-identified and authorized entities must be able to access the management data and thus be given the right credentials to perform some operations. Our approach is driven by an efficiency effort with a large P2P type functionality, where peers can perform management operations on other peers.

The secondary contribution of our paper is an instanciation of the integrated security framework to the recently proposed Netconf protocol [11]. Several XML-based management solutions have emerged over the past few years. XML-native approaches like Junoscript and Netconf or hybrid approaches like XML/SNMP gateways are representative of an effort to use XML for network management. XML, whose main quality is to be both human and machine-readable, makes it possible to model and organize data hierarchically, to describe data in a formal way with XML.
Schemas, to exchange easily data over different platforms and to handle data efficiently with a very large panel of available tools (XSLT, SAX, DOM). Moreover XML now integrates security features like digital signature [5] or encryption [14]. This allows the definition of customizable, fine-grained, i.e. at XML element level, security policies. Network management did not address and propose a security architecture for XML-based management yet. A security architecture is expected to provide authentication, confidentiality and access control while remaining scalable and being integrable in existing modules.

Examples of a multi-provider management context can be either a distributed firewall configuration, where one firewall protecting the internal network can push a set of rules onto the frontal firewall dynamically when it receives abnormal traffic. This process can be reversed if the frontal firewall wants to configure the filtering rules of the internal firewall. A multi-homed environment is another example of a natural multi-actors interaction for management purpose where flexible security is required. These actors can be bound to specific roles and management responsibilities [18]: security configuration, fault management, performance management.

To cope with these scenarios, we present in the rest of this paper our solution. To this end, the paper is organized as follows. Section 2 presents the context and the addressed issues. Section 3 describes the overall architecture. Section 4 describes the implementation of our security framework. Related work is addressed in section 5. Summary and future work are provided in the conclusion.

2. Context

Network management becomes a complex task within large scale networks, where several thousand devices are managed by large groups of network/service managers. As a consequence, the amount of people in charge of maintaining this network also grows significantly. The difficulty to make all the devices cooperate to provide an acceptable quality of service is linked to the ability to define network management tasks, while assuring the necessary security features to allow authorized principals to perform management operations. The way to organize people within enterprise functions (or roles) can be addressed using the well-known Role Based Access Control model (RBAC [15]). This model addresses scalability issues in terms of number of users, since they are organized into roles which in turn are organized into an hierarchy. Scalability, when implementing an RBAC model is important. Roles can be associated to users in a dynamic and distributed way.

Performing a request under a role previously requires an authentication mechanism. A user can authenticate himself on all managed devices which implies heavy Public Key Infrastructure (PKI [1]). An alternative is for a user to authenticate himself to a server in order to obtain some credentials first and then is able to authenticate on agents using these credentials. Moreover, instead of using user-centric credentials, these credentials can be role-based so that a user authenticates indirectly himself on agents through a role. Thus, agents are only able to authenticate roles, consequently decreasing the complexity of access control policy within the agents, since the number of roles is far smaller than the number of users in a large network. Such a mechanism using keys for authentication as well as for encryption requires an efficient organization to distribute and refresh keys, in particular in a context where users can have multiple functions at different time.
Some performant algorithms have already been proposed in a multicast security context and can be reused as demonstrated in this paper.

Assuming a promising XML-based network management, we propose in this paper a security framework for network management with performance and scalability requirements, making use of the previously described concepts. We instantiate this within a Netconf support [11].

Such a protocol introduces security issues and therefore, only authorized principals must be able to set up securely network devices configurations. According to the Netconf specification, the application protocol on which Netconf lays is responsible for providing authentication, data integrity, and privacy services [11]. We consider that security should be integrated within Netconf allowing an enterprise-level integrated security framework and therefore propose an alternative solution.

3. Integrated Security Framework

Figure 1 shows our integrated security components. The core is split into four main security areas: authentication, integrity, confidentiality and access control. Access control means a flexible mechanism to manage access to resources. Flexible aspect is evaluated regarding the platform dynamicity and size. While dynamicity implies the capability to frequently modify the access rights, the size is related to the amount of both devices and managers. Authentication is the mechanism that guarantees that an entity is the one it claims to be. Integrity is the process that warrants that the management data in transit has not been modified.

In order to address this issue, we conceive a distributed Role Based Access Control (RBAC [15]) 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. Therefore, we propose the use of already defined XML security paradigms (XML-Signature [5], XML-Encryption [14]) to perform data access control.

Architecture

Our secure model for Network configuration is architectured around several main entities depicted on Figure 1. 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 configuration 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 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: authentica-
Figure 1. Conceptual security components

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 authorization policies while reducing errors and administration costs. Introduction of administrative roles and role hierarchies made it possible to considerably reduce the amount of associations representing permissions to users allocation. In this paper, we consider the NIST (National Institute of Standards and Technologies) RBAC model [15] which gathers the most commonly admitted ideas and experience of previous RBAC models.

Our extension proposes a distributed RBAC model, where roles are dynamically deployed and cryptographic mechanisms are used in order to provide efficient and scalable authentication and confidentiality services. In order to perform authentication of entities to the RBAC manager, the RBAC manager communicates with a Public Key Infrastructure (PKI) [1] which delivers certificates to both configuration providers and managed devices. Thus, we assume that all of them (configuration provider and managed device) own a public/private key pair to provide strong authentication. An entity authenticates itself on the RBAC manager by sending a message signed with its private key. Once the entity has been authenticated, the RBAC manager can distribute the keys for a particular role queried by the entity. These keys will be used to perform network operations directly on agents. This is equivalent to a Single Sign-On process where the credentials distributed by the authenticator to access to the service (the agent) are the role keys.

We propose to bind each RBAC conceptual role to a specific key. This means that each entity who wants to perform an operation in the context of a role needs to upload this key. The RBAC manager is responsible for distributing these keys to the different entities. This way, only the
entities who possess the role key are able to decrypt and encrypt the message for a particular role. Indeed, when an entity is not entitled any more to a given role, it should not be allowed to perform operations endorsed by this role. Therefore, efficient key management (distribution) is required, in order to dynamically re-distribute a new key to the entities endorsing a given role. We call this set of entities a $KeyGroup_{role_i}$.

We give now some definitions to formally describe the credentials and sets used in our architecture:

- $E$: the set of entities of our architecture,
- $k_e^{public}/k_e^{private}$ where $e \in E$: denotes the public and private keys owned by device $e$,
- $roles(e)$ where $e \in E$: the set of roles that appears at least one time in the ACL rules of the entity $e$. This represents all the keys needed by an agent to decrypt and encrypt data for a role. An ACL is bound to each node of the XML configuration in order to allow dynamic access control. An ACL is an XML node containing a set of rules describing a set of available operations for a given role. $roles(e)$ is built by parsing all the ACL of the XML document configuration in an agent and adding each appearing role in this set. We define the ACL XML syntax in section 1.3.0.0. The set $roles(e)$ is used to define the set of keys needed by an entity, each role being linked to a key,
- $k_{role_i} = \{k_{role_i}^{priv}, k_{role_i}^{auth}\}$: symmetric cryptographic key corresponding to the role $role_i$ and shared by all the entities that currently endorse $role_i$. Each role uses two keys: $k_{role_i}^{priv}$ is used to encrypt XML messages sent on $role_i$’s behalf while $k_{role_i}^{auth}$ is used to authenticate them. Each entity can potentially send messages using these keys, provided that it is allowed by the RBAC manager to activate this role. The use of these two keys is similar to what is done in User Security Model [6] of SNMP. One key is used for authentication, while the second key is used for encryption. However, a key is bound to a user in SNMP, whereas a key is bound to a role in our approach,
- $KeyPath_{role_i}$: a path in a key tree. It contains the set of cryptographic keys for $role_i$ needed by $e$ for key refreshment. The $KeyPath$ for entity $d$ is depicted in grey in the second tree of Figure 2. $d$ has endorsed the role $role_A$. The root key is used to encrypt data when a message is sent under the corresponding role. The leaf keys are known only by the submember and are sent using the PKI. The intermediate keys are used to send the new keys in an efficient and secure way. We describe the rekeying mechanism in section 1.3.0. The set $KeyPath_{role_A}$ is equal to the keys set $\{kad, kcd, kd\}$. While $kd$ is the initial key specific to $d$, obtained with the PKI, $kad$ is equal to $k_{role_A}$ and $kcd$ is an intermediate key known by $c$ and $d$,
- $KeyGroup_{role_i} = \{e \in E | role_i \in roles(e)\}$: set of devices which belongs to $role_i$.

The RBAC manager also interacts with a key manager. The latter manages different key trees (see Figure 2) whose keys are distributed to the configuration providers and managed devices using
IP multicast in order to ensure efficient key distribution. All the agents and managers are members of the same multicast group so that they all receive the key refreshment messages.

A key tree is bound to a group. The leaves of a tree correspond to a particular entity. An entity must be advertised of all the keys from its leaf to the root so that key refreshment is possible. The PKI is used to exchange the initial (leaf) key of an entity. Once this key has been received by an entity, the public/private key are no longer used in an effort to optimize the efficiency of our secure architecture. We discuss this at the end of this section. Finally, an RBAC configurator sets the access control policy. RBAC manager maintains associations linking users and roles. When an entity asks for a role, the RBAC manager looks up its user-to-role database to decide if the user can endorse the role.

In fact, a key tree is used for each role in order to provide an efficient key refreshment. This makes it possible to refresh the $k_{role_i}$ of the entities of $KeyGroup_{role_i}$ without using their public/private keys. In order to refresh the key group in an efficient way, we use logical key hierarchy (LKH) [24, 26, 25] algorithm to update a minimal set of keys in a multicast way.

We now describe the interactions between the configuration providers, the managed devices and the RBAC manager:

- When a configuration provider wants to perform some operations on a managed devices, it must first endorse a role. In order to be given a role, a configuration provider must send a request to the RBAC manager. This request contains the roleID of the desired role and the identity of the sender to retrieve its public key. It is encrypted with the public key of the RBAC manager so that nobody else but the RBAC manager can decrypt the message and authenticated with the private key of the configuration provider so that a receiving entity is sure that $e$ is the message originator. Formally, the message to be sent to the RBAC manager in order to apply for a role roleID is the following:

$$k_{RBACmanager}^{RBAC}(roleID, identity, k_{private}^{e}(roleID, identity))$$
This is possible thanks to the PKI which delivers certificates to both the configuration provider and managed devices,

- If the message is valid, the RBAC manager looks for a valid entry in its user-to-role associations set. If the asked role (roleID) can be endorsed by this user, the RBAC manager queries the key manager to get the different keys linked to this particular group. The RBAC manager must send $KeyPath_{role_i}^e$ securely to the configuration provider who wants to activate role roleID. Therefore $ke$ (the leaf key of entity $e$) is hashed and signed with the private key of the RBAC manager so that the receiving entity is sure that the sender is the RBAC manager. Moreover, the message is encrypted with the public key of the receiver so that nobody else but the receiver can decrypt the message:

$$k_{public}(ke, k_{RBACmanager}^{private}(ke))$$

Then, $KeyPath_{role_i}^e$ is sent using $ke$ secret. Thus, we assure authentication and confidentiality services for the KeyPath transmission:

$$ke(KeyPath_{role_i}^e)$$

More details on key management are given in the section 1.3.0,

- Once a role has been endorsed, a Netconf manager is able to send secure messages to all devices. Secure here means both encryption, authentication and integrity with the role keys $k_{role_i}$. The agents also have an access control system to limit the operation a role is allowed to perform. Formally, a request sent from an entity acting as a manager to an agent under a role role$_i$ is the following:

$$k_{role_i}(<rpc>::=<rpc>...)$$

These services are described in the next section,

- When an entity drops an active role, the RBAC manager informs the key manager to remove the entity from the corresponding tree. All the keys from the entity key to the root of the tree must be refreshed so that the entity can no longer perform operations on the managed devices.

**Security extension to the Netconf protocol**

**Authentication, integrity and confidentiality.** In order to secure Netconf, managers and agents need to have a shared secret. In order to perform both authentication and encryption services, we need two keys. Once a manager has been authenticated to the RBAC manager, it communicates with the agent having the rights of a role that he endorsed beforehand. The manager owns the two keys $k_{priv}^{role}$ and $k_{auth}^{role}$ for this role. Each Netconf message is sent in the context of a role, not a user. This makes it easier for both agents to perform access control and key manager to distribute and refresh keys.
Each Netconf message is encrypted with the key $k_{\text{priv}}^{\text{role}}$ and stored in an `<EncryptedData>` element in accordance with the W3C [9] recommendation XML-Encryption [14].

Concerning the authentication service, an XML element is then signed using the key $k_{\text{auth}}^{\text{role}}$. The result is a `<Signature>` element in accordance with the W3C recommendation XML Digital Signature (XML-DS [5]). It contains the description of the algorithms used to build the signature as well as the information to retrieve the credentials that must be used to verify the signature (certificates, keys, ...). Figure 3 illustrates the XML encapsulation using XML authentication and encryption process.

![Figure 3. Netconf message encapsulation](image)

**Access control.** Many different access control models have been proposed to protect XML data. Most of them are role-based [15, 4]. These models are recognized to be scalable thanks to different properties like the role concept by itself and role hierarchies. However, it most often uses an external formalization with XPath references to the protected document. It remains difficult for an access control administrator to see what is currently accessible or not for a role because he has to parse all the permissions. We propose an XML integrated role-based Access Control List (ACL) mechanism. Each node of the XML document representing the device configuration is protected by a set of access control rules. In concrete terms, a rule describes an operation a role is allowed to perform. We organize the Netconf operations in terms of `<get>`, `<merge>`, `<replace>`, `<create>` and `<delete>` basic primitives as depicted on Table 1. While the Netconf operations stand on the left column, the basic operation are placed on the right. When a Netconf primitive is requested on a node, the access control process consists in checking that the corresponding operations are in the ACL. The fact that a role is allowed to perform a particular primitive on a node does not mean that it is allowed to perform the same primitive on the children or parent nodes. This greatly simplifies the process of access control and gives a fine granularity level. Each node is responsible for its own access control. However, an ACL editor application may give access to all parent nodes automatically when giving access to a particular node depending on the preferences of the rule editor.

Figure 4 shows how an access control list is added to an XML node. It defines that `myFirstRole` can get this node, replace this node and merge this node but does not subsume anything for the
An Integrated Security Framework for XML based Management

<table>
<thead>
<tr>
<th>Netconf operations</th>
<th>ACL basic primitives</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;get-config&gt;</td>
<td>&lt;get&gt;</td>
</tr>
<tr>
<td>&lt;edit-config&gt; (operation=merge)</td>
<td>&lt;merge&gt;</td>
</tr>
<tr>
<td>&lt;edit-config&gt; (operation=replace)</td>
<td>&lt;replace&gt;</td>
</tr>
<tr>
<td>&lt;edit-config&gt; (operation=create)</td>
<td>&lt;create&gt;</td>
</tr>
<tr>
<td>&lt;edit-config&gt; (operation=delete)</td>
<td>&lt;delete&gt;</td>
</tr>
<tr>
<td>&lt;copy-config&gt;</td>
<td>&lt;delete&gt;</td>
</tr>
<tr>
<td>and &lt;replace&gt;</td>
<td>and &lt;create&gt;</td>
</tr>
</tbody>
</table>

Table 1. Netconf operations to basic primitives mapping

parent and children nodes. The second role entry states that the role mySecondRole can only get this node. The fact that we use roles considerably limits the amount of <rule> tags in the ACL.

```xml
<interface>
  <ACL>
    <role roleRef="myFirstRole">
      <operations><get/><replace/><merge/></operations>
    </role>
    <role roleRef="mySecondRole">
      <operations><get/></operations>
    </role>
  </ACL>
  <name>Ethernet0/0</name>
  <mtu>1500</mtu>
</interface>
```

Figure 4. An ACL for a particular node

Using this model, it is possible to build a Netconf XML response depending on the access rights of a role. First, in the case of a Read operation, the resulting XML document is built without taking care of the access control rules. This gives a subtree of the whole configuration tree. Next, the subtrees for which the role can not perform the requested operation are pruned. The next step consists in encrypting the tree with the key corresponding to the role in use: $k_{role}^{priv}$. Once this is done, an XML-Signature is built using $k_{role}^{auth}$. In the case of a Write operation, there is no need to build an XML document but additional verification assures that the role is allowed to write in the requested node(s).

We assume that we trust the authenticated entities meaning that these entities, having some role, will not use their privilege (their role keys) to spoof another entity having the same role. In particular, we assume that an entity will not intercept a request and send back a response in place of another agent.
Key management

The Key Manager is the entity responsible for managing key trees for each group. We bind a key tree to each role, the root key serving as the role key. Since several entities can endorse the same role simultaneously, all of them must receive the key for that role. Revocation of a role to a member can happen dynamically so a key refreshment mechanism is needed to deter a newly arrived member from decrypting past messages and past member from de(en)crypting new messages.

Several rekeying algorithms have already been proposed among which Marks [8], LKH [24, 26, 25] and One-way function tree (OFT). We propose to use LKH which best suits to our requirements. Marks implies that the time membership is known at the beginning which is not possible in our case. LKH+ proposes a rekeying algorithm with a log(n) complexity for both join and leave rekeying. The key group (key role in our case) is the root node of the tree. Each member is a leaf and knows all the keys from its own key to the root. Its own key is distributed by the Key Manager using the public/private key pair. The other keys are sent in the same way as initializing of LKH. An important aspect in this algorithm is that a member must know all the keys from its own key to the root. We previously called this key set a KeyPath.

Let’s illustrate the use of LKH approach to manage a member join or leave messages. Figure 2 displays three trees corresponding to the same role: the first one is the initial key tree and contains three members \( \{a, b, c\} \). If a new entity \( d \) wants to join the role, the key tree must be updated as efficiently as possible. \( d \) must be given the root key to be able to handle future messages. This root key is changed so that \( d \) can not read old messages. We use here a PKI to communicate the symmetric key \( k_d \) for entity \( d \) encrypted with its public key. He is the only one who can decrypt this initial message. Then the other keys of the set \( \text{KeyPath}_d^A \) are sent to \( d \) and to the other entities as depicted on the figure. The transition from the second to the third tree illustrates a leave rekeying. Entity \( a \) knows \( k_a \), \( k_{ab} \) and \( k_{ad} \). Since \( k_a \) and \( k_{ab} \) disappeared from the tree when \( a \) leaves the role, only \( k_{ad} \) should be updated so that \( a \) can not access future traffic. Therefore the key manager generates a new key \( k_{bd} \) and sends it to all the entities under this key, namely \( b, c \) and \( d \) in such a way that the number of message is minimal. For instance, a single message is needed to send \( k_{bd} \) to both \( c \) and \( d \) entities as they both know \( k_{cd} \). The two messages are given under the third tree.

We use these trees in order not to encrypt new role keys with the public keys of the entities, since this would be to expensive.

4. Implementation and design

We did a first implementation of the framework on top of Yenca [28], an open-source Netconf implementation realized by our research team. Figure 5 depicts the architecture implemented in our prototype. A Netconf manager is depicted at the top of the figure. It communicates with the Key Manager to obtain a role and the corresponding keys which will be used later to perform configuration operations on the agent. A Netconf agent is illustrated at the bottom of the figure. This managed device embeds a security manager to be able to communicate with the managers in
An Integrated Security Framework for XML based Management

...a secure way. It consists in three main parts: the core in charge of parsing and handling messages, the security manager and the modules set.

The entity acting as a manager is represented at the top of the figure. It consists of three layers. The top layer provides a graphical tree representation of the XML configuration data for the administrator. The intermediate layer is a Document Object Model (DOM [23]) parser that is able to build a DOM structure from the received XML data or to serialize requests from the administrator. The bottom layer is the communication layer which is in charge of sending and receiving Netconf messages from the network. The manager also contains a database that contains all the keys needed depending on all the roles a manager has endorsed.

Figure 5. Netconf secure architecture

The agent is represented at the bottom of the figure. The Networking layer is used to emit and receive Netconf message over the network. The parser layer is in charge of extracting the different parts of the Netconf messages among which the target configuration and the operation. Then the request dispatcher forwards the request to the right module depending on the name of the root...
tag element under <config>. For instance, the message on the left of Figure 5 is handled by the interface module.

The core part of the agent is extended with a security module providing a set of services:

- managing ACL rules: The module is able to parse dynamically the ACL embedded in the data configuration tree. When a request is received, the resulting document is built firstly without taking care of the ACL rules. Then the module is called to prune the subtrees which are not available for that role, operation. It is also possible to validate the ACL rules,

- authenticating and encrypting messages: The module retrieves the key $k_{role_i}$ which is the key corresponding to role $role_i$. Then the XML document is encrypted with this key so that only the entities from the set $KeyGroup_i$ can decrypt the message. The encrypted document is valid regarding the XML-Encryption recommendation,

- managing keys dynamically: This service is responsible for managing KeyPath coming from the RBAC manager.

We now detail the main security related interactions of the process on Figure 5:

- First, interaction 1 depicts how a module registers to the Module subscription handler. The interfaces module informs the module subscription handler that it is responsible for all requests containing the <interfaces> element as root. The root node under the <config> element must be the same as the name of a Netconf module,

- In Interaction 2, the Key Manager sends the KeyPaths to each Netconf entity e for all their available roles: $\forall role \in roles(e), KeyPath_{role}^e$. Once the two previous interactions are performed, it is possible to send secure requests,

- Interaction 3 shows the level in which we use the key corresponding to a role to encrypt and authenticate data,

- Interaction 4 shows the message transmitted over the network. It is embedded into a <Signature> element after being encrypted into an <EncryptedData> element,

- Interaction 5 depicts the interaction with the Security Manager to decrypt and authenticate the incoming message. The role to use is added in the KeyInfo element of the XML Signature,

- Interaction 6 shows how the request dispatcher delegates the handling of the request to the module on the base of the data contained in the request,

- In interaction 7, the security manager receives the XML response containing ACLs from the Request dispatcher and uses these ACLs to prune the XML response. This mechanisms depends on the current Netconf operation and the role in use,

- Once the document is reformatted, it is encrypted and signed in interaction 8 in the same way as during interaction 3,
Interaction 9 shows the transport over the network to the destination,

Interaction 10 emphasizes the decryption and validation process on the manager side.

All of the Netconf’s modules must respect the structure representation that is used by the Yenca core to represents a generic module. The module interface defines several generic function that must be implemented by each module. The register function allows the module to register to the Module Subscription handler. The module implements the get_request function that makes use of setcfg and getcfg to answer Netconf requests like <get-config> and <edit-config> depending on the request structure. A Netconf module must only be compliant with the generic interface described just above.

5. Related work

In an XML environment, the first network management protocol to propose security mechanisms was Junoscript [19]. 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. One advantage is that the same request can be send very easily to many devices with the same encryption and authentication since it is sent under a role. This is not possible with SSH where a session must be open on each device which is not scalable.

Three parameters enter into account for scalability issues: network size, number of network managers and dynamicity of roles (activation and revocation). Our cryptographic process integrated within an RBAC model and our P2P approach fulfills simultaneously these three requirements. Network size is addressed by the P2P network management architecture, managers are bound to roles thus decreasing dramatically the number of keys and access control rules and we use key refreshment to deal with role administration.

Related work to SNMP based security approaches is given in [21], where we propose a policy based access control extension to the SNMP stack of an agent.

In [13], the authors propose a new security model for SNMPv3 whose main goal is to reuse existing security mechanisms like PKI or Radius. This approach is based on sessions and allows to build keys dynamically contrary to what is done in the default security model, USM.

Some intermediate solutions for XML management were proposed with XML/SNMP gateways [22] [20]. These gateways allow managers to query their SNMP agents without knowing about SNMP. This provides a quite easy 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, keys distribution
and access control policies. In [10], we extend these gateways with security features and provides an end-to-end security continuum.

Since we also address distributed XML applications, we use the XML security framework ([5, 14, 7]) to protect Netconf messages. However, we do not use SOAP encapsulation contrary to Web Services.

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] [3]. A similar approach has been submitted recently in the Netconf mailing list [2].

Lupu et al. presented in [18] [16], [17] 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.

6. Conclusion

In this paper, we introduced an integrated security framework for XML-based network management. This framework provides different security services (authentication, integrity, confidentiality and access control) and addresses scalability and dynamicity requirements. It is based on Single Sign-On mechanism, where a manager first requests a role activation and then receives the credentials from the RBAC server. An underlying PKI is used to initialize the role activation. These credentials (role keys) are used to authenticate and cipher configuration operation requests to the agents. Access control is performed on the agent since each XML node of its configuration contains ACL bound to roles.

Scalability issues are addressed at different level: role concept allows to aggregate rights to management functions thus decreasing the number of ACL; the RBAC model is distributed on the agents thus avoiding a bottleneck central access control server; role key are distributed and refreshed in an efficient way by using IP multicast and specific algorithms based on keys trees.

Dynamicity is addressed thanks to role endorsement and revocation. In these two cases, the role keys have to be updated as fast as possible so that the revoked manager can no longer access future operations and the new manager can not access past operations.

Acknowledgment. We would like to thank Isabelle Chrisment for the discussions on protocols for large groups key exchange.
References


