Firewall Configuration Management Using XACML Policies

Tugkan Tuglular, Fusun Cetin, Oguz Yarimtepe and Gurcan Gercek
Department of Computer Engineering
Izmir Institute of Technology
Gulbahce Koyu, Urla, Izmir, Turkey
Telephone: +90-232-7507875
Fax: +90-232-7507862
Email: {tugkantuglular, fusuncetin}@iyte.edu.tr, {oguzyarimtepe, gurcangercek}@std.iyte.edu.tr

***Abstract*—This paper proposes an architecture for XACML based management of firewall configurations in large enterprise networks. The goal of this architecture is to allow administrators and end-users to manage their firewalls, while enforcement of organizational policy is ensured to prevent unacceptable traffic gaining access to the private network domain. The central architectural component is the domain policy server which pushes organizational policy down to firewalls deployed in its domain. In addition to its reporting function, the domain policy server monitors and verifies policy changes, i.e. checks for inter- and intra-firewall anomalies, on any firewall within its domain. The proposed architecture includes firewall agent components, where one resides on each firewall, through which coordinated operations on firewall policies are achievable. Firewall policies, topologies, and configuration messages that are stored and exchanged within the architecture are presented in XML. Although available XACML is used for the representation of firewall policies, two DTDs are developed to express topologies and configuration messages. A prototype implementation of this architecture is presented in this paper along with examples of firewall configuration management operations.**

I. Introduction

Maintaining security on their networks is critical for all enterprises. One primary security mechanism that every network needs is access control - the ability to carefully define and enforce which users have what type of access to specific applications, data and services [8]. When the network is contained within a single building, the problem is generally handled by a single firewall. For enterprise networks, which involve multiple interconnected segments distributed across various locations, the problem is hard and beyond node level configuration. To manage this kind of distributed networks, system administrators want to focus on organizational policy level management instead of spending their time for node level configuration [15]. Existing approaches and tools are inadequate to meet the current demands of distributed installations requiring frequent and error-prone reconfigurations [4].

This work proposes an architecture for the management of firewall configurations in networks that span multiple subdomains, or subnets, which may be administered by different people with different views of network security. One of the specific goals of this work is to verify that the local firewall configurations enforce organizational policy and there is no anomaly between organizational and local policies. So that the paradigmatic example - even if each of two connected firewalls correctly implements the local policy, the interconnection of the two firewalls may violate organizational policy that neither firewall can detect by itself [4] - shall not be a concern for the administrators anymore.

This paper provides two contributions. The first contribution is to introduce a management architecture for firewalls, which are distributed over the domain of a large enterprise network. Management functions, such as policy providing, anomaly checking, change management and reporting are collected on a policy server. Moreover, the proposed architecture includes firewall agents residing on managed firewalls, wrapping their functionality.

The second contribution of this paper is to utilize XACML for the representation of firewall policies. In addition to firewall policies, topology information stored in the policy server and configuration messages exchanged between policy server and firewalls are also XML based. XML based technologies are increasingly utilized in the management of network security devices, which is a complex task for large enterprise networks, where several devices are managed by network or security administrators. We selected XACML over WS-Policy because previous works [3] and [14] point out that WS-Policy has some weak points, or shortcomings, when used to express access control policy.

The paper is organized as follows: an overview of related work is provided in the next section, which is followed by the description of the proposed architecture and its components. Then XACML representation of firewall policies as well as XML representation of topology information and configuration messages is explained with examples. The next section presents a prototype implementation of this architecture, which is followed by the presentation and discussion of the experimental values obtained by running the prototype implementation. The last section summarizes our findings and contributions as well as describing future work.

II. Related Work

Several research topics are covered in this paper, which are eXtensible Access Control Markup Language (XACML), policy anomaly checking and policy based management of firewalls. In this section we briefly explain these topics.

XACML is an OASIS standard that describes both a policy language and an access control decision request/response language, which are both encoded in XML. The policy language is used to describe general access control requirements and has standard extension points for defining new functions, data types and combining logic [13]. Everything described in the XACML version 1 specifications is supported in library developed by SUN.

In our proposed architecture we used the policy anomaly detection algorithms for intra- and inter-firewall policy anomalies described in [2]. In order to analyze the firewall policy anomalies, modelling of firewall rule relations is necessary. In this work, the model which is created by [1] is used to analyze firewall policies. They defined all possible relations between filtering rules and they proved that no other relation exists. Using these relations they defined anomalies that can be found in a firewall (intra-firewall) and between firewalls (inter-firewall) and developed algorithms to detect the defined anomalies. These algorithms are implemented and results are presented in [5].

A significant number of studies can be found in the literature on policy based management of firewall configurations. For instance, one of the studies demonstrates that use of XML technology as a middle layer provides a means to combine the security of trust management system with the simplicity of a policy editor [11]. Another paper proposes an integrated security framework for XML based management in large enterprise networks [6]. There are other efforts, such as [7] and [10], which define policy based management operations to be implemented in architectures like ours.

III. Firewall Configuration Management Architecture

Before we describe the main components and services of our proposed architecture, it is necessary to present how we modelled a network domain. As illustrated in Fig. 1, at the domain premise there exist a domain firewall, which protects the entire domain with respect to organizational policy. The number of subnets connected to the domain firewall is up to the organization, but there should be at least one DMZ where the management server of our architecture exists. The only requirement for all subnets is that the network connections follow a tree structure, so that the entire domain can be modelled as a tree where each node contains a firewall. The root node will be the domain firewall, the subnet firewalls will exist at the intermediate nodes, and personal firewalls will be at the leaves of the topology tree.

Two main components of our architecture are Domain Policy Server (DPS) and Firewall Agent (FA). The components of DPS are shown in Fig. 2. DPS is composed of firewall policy manager, organizational policy provider, policy change manager, policy anomaly checker, policy report manager and policy repository components. The components of DPS may be distributed over the DMZ, domain or Internet.

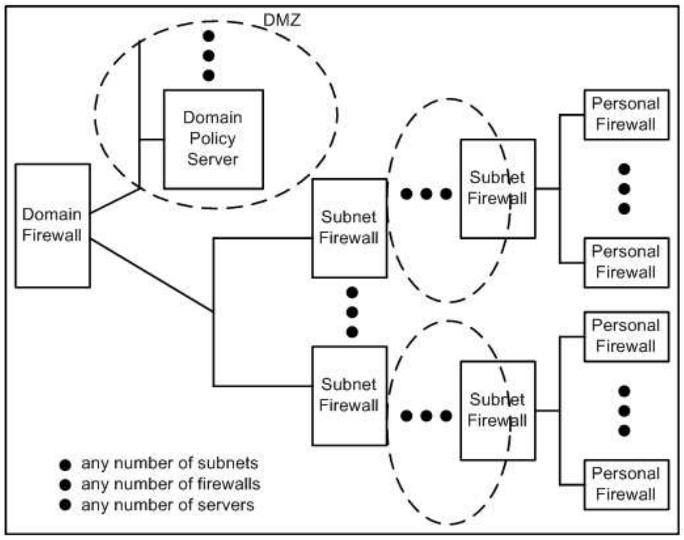


Fig. 1. Network Domain Model

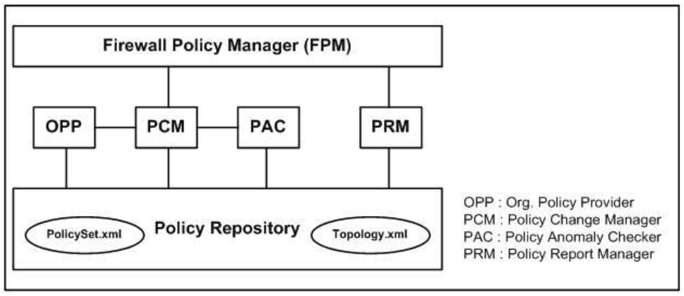

Firewall Policy Manager (FPM)
OPP
PCM
PAC
PRM
PolicySet.xml
Policy Repository
Topology.xml
OPP : Org. Policy Provider
PCM : Policy Change Manager
PAC : Policy Anomaly Checker
PRM : Policy Report Manager

Fig. 2. Firewall configuration management architecture

Firewall policy manager handles policy operations, which are executed through the interactions among administrators, FAs, and DPS components. These operations are defined as follows:

- Maintain policy rules for all firewalls in the domain and their topology information
- Present firewall policy rules in an implementation independent way
- Manage organizational policy rules
- Distribute organizational policy rules to firewalls deployed in the domain
- Monitor firewall configuration changes, i.e. deployment of a new firewall to domain network, removal of an existing firewall from domain network, addition of a new rule to the firewall policy, and deletion of an existing rule from the firewall policy
- Check firewall configuration changes for inconsistencies with respect to organizational policy and other related firewall policies
- Convert implementation independent organizational policy rules into Iptables firewall rules
- Exchange configuration messages to maintain order in the network domain

Organizational policy provider is responsible for the distribution of organizational policy to FAs. If there is a change in the organizational policy this change is propagated to all FAs within the domain under the control of policy change manager. The policy change manager also handles configuration change requests coming from firewalls through the FAs. These requests are first sent to policy anomaly checker to ensure that rule to be added or deleted is not in conflict or inconsistent with the organizational policy as well as with the policies enforced enroute to domain firewall. The route or path information is obtained from Topology.xml file and concerned policies are retrieved from PolicySet.xml file. These two files are stored and maintained by the policy repository. The policy report manager component is used to present policy and topology information on the user interface.

FA is responsible for translation of implementation independent rules to Iptables commands and for their execution so that organizational policy is installed and enforced by the local firewall which is either subnet firewall or personal firewall. Subnet and personal firewalls are expected to enforce both the organizational policy and its local policy which should be consistent with policies of other firewalls that are in the path to the domain firewall. This architecture assumes that the FA is installed along with Iptables firewall.

As mentioned above, the DPS and FAs exchange configuration messages. We organized policy management functions in terms of basic GET, ADD, DEL commands as depicted in Table I. The commands generated by either party will be transformed into XML and sent to the other party. As can be followed from the direction column in Table I, requests from FA to DPS may only be ADD and DEL commands, whereas requests from DPS to FA may include a GET command, which can be used to check whether previous ADD or DEL command is executed as expected.

TABLE I
SUMMARY OF CONFIGURATION MESSAGES

Direction	Command	Parameter	Explanation
FA->DPS	ADD	ordered-rule	add ordered-rule
FA->DPS	DEL	i	delete ith rule
DPS->FA	ADD	ordered-rule	add ordered-rule
DPS->FA	DEL	i	delete ith rule
DPS->FA	GET	i	get ith rule

IV. Representation of Firewall Configuration Management Information

Information stored in DPS and information exchanged between DPS and FA is represented in XML. The underlying XML representation is made transparent to the administrator and end-user, so that various views can be built on top of the represented data, which is an ideal attribute for an enterprise that needs to manage distributed firewall policies [12]. In this section, the information structures for firewall policies, network topology with respect to firewalls and configuration messages are explained and their XML representation is presented.

A. Representation of Firewall Policies

According to previous work [1], the following is the common format of packet filtering rules in a firewall policy:

$$< order, protocol, src\_ip, src\_port, dst\_ip, dst\_port, action >$$

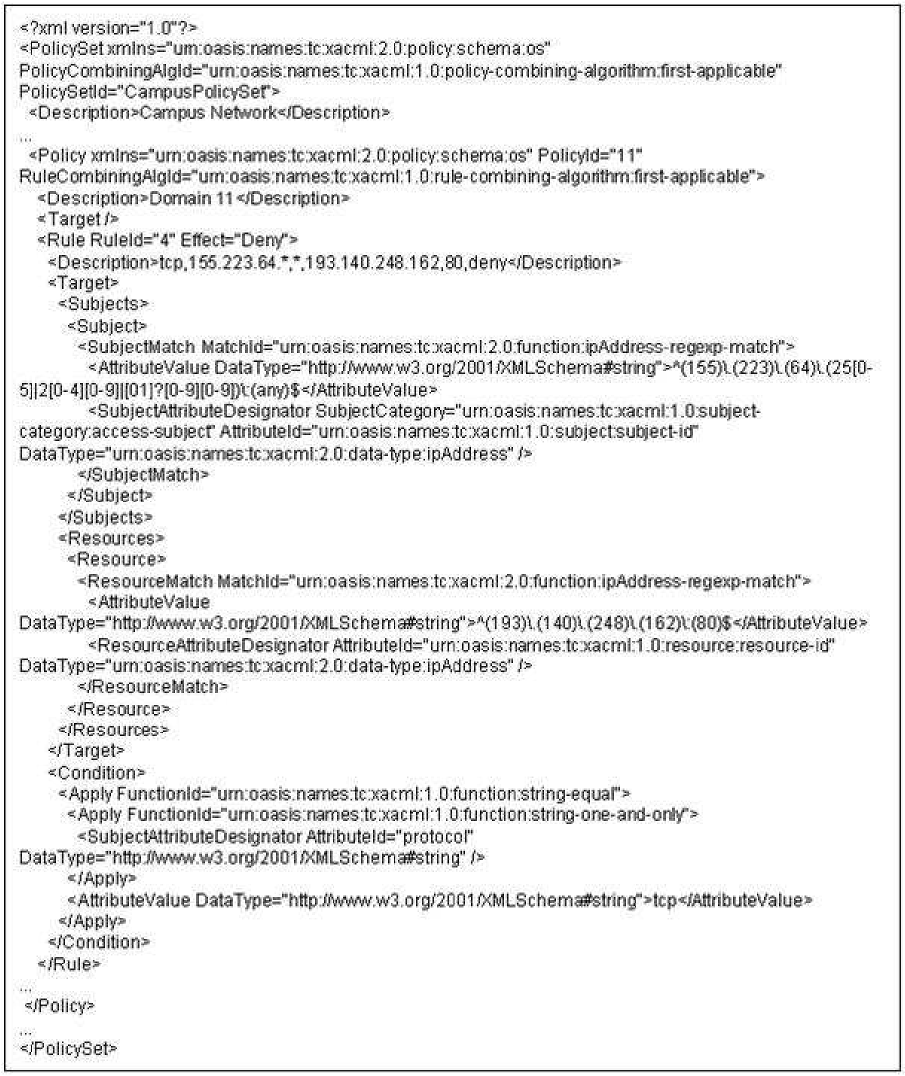
In this paper, we refer to this format as ”7-tuple ordered-rule”, or simply ”ordered-rule tuple”. Similarly, a ”packet tuple” is <protocol, src_ip , src_port, dst_ip, dst_port> and a ”rule tuple” is <protocol, src_ip, src_port, dst_ip, dst_port, action>. As explained in [1];

- The order of the rule determines its position relative to other filtering rules
- The protocol specifies the transport protocol of the packet
- The src_ip and dst_ip specify the IP addresses of the source and destination of the packet respectively
- The src_port and dst_port fields specify the port address of the source and destination of the packet respectively
- The action is the operation to be performed after a match

Use of ordered-rule tuple enables us to have an implementation independent firewall policy rule representation. However, firewall policy representation for the management of distributed firewalls requires more information to be handled. Therefore, we chose XACML as the policy description language. In XACML, policy element represents a policy for a firewall whereas policy set element represents policies of a set of firewalls existing in a domain. This way, we are able to represent relations among firewall policies and to embed other information related to a firewall into its policy such as; firewall IP address, firewall agent number, implementation independent representation of rules and rule combining algorithms.

We developed a method to represent distributed firewall policies in a single XACML policy file. The XACML PolicySet element holds all the policies of firewalls deployed in the domain. Organizational policy is stored as the domain firewall policy. The XACML Policy element encapsulates all the rules of a firewall policy. A XACML Rule element represents a single rule of a firewall policy, which means an ordered-rule tuple is encapsulated in an XACML Rule element. The order and action fields of the tuple are represented as the attributes of XACML Rule element. The protocol field of the tuple is stored in the XACML Rule Condition element. The src_ip and src_port fields are held in a Subject element of the XACML Rule Target element. Similarly, the dst_ip and dst_port fields are held in a Resource element of the XACML Rule Target element. The implementation independent representation of a rule is stored in the XACML Rule Description element.

A part of the PolicySet.xml file is illustrated in Fig. 3. The part taken from the file shows the XACML representation of the rule number 4 of the firewall numbered as 11. Firewall 11 is illustrated in Fig. 5, which shows the graphical user interface of our prototype implementation. Examples of the XACML elements explained above can be found in Fig. 3. Furthermore, it is important to note that IP addresses and port numbers are represented with regular expressions.



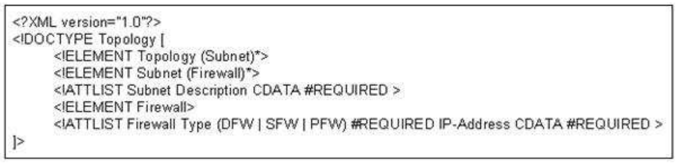
```
<?xml version="1.0"?>
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable"
PolicySetId="CampusPolicySet">
  <Description>Campus Network</Description>
...
  <Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os" PolicyId="11"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable">
    <Description>Domain 11</Description>
    <Target />
    <Rule RuleId="4" Effect="Deny">
      <Description>tcp,155.223.64.*,*,193.140.248.162,80,deny</Description>
      <Target>
        <Subjects>
          <Subject>
            <SubjectMatch MatchId="urn:oasis:names:tc:xacml:2.0:function:ipAddress-regexp-match">
              <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">^(155)\.(223)\.(64)\.(25[0-
5]|2[0-4][0-9]|[01]?[0-9][0-9])\:(any)$</AttributeValue>
              <SubjectAttributeDesignator SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-
category:access-subject" AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="urn:oasis:names:tc:xacml:2.0:data-type:ipAddress" />
            </SubjectMatch>
          </Subject>
        </Subjects>
        <Resources>
          <Resource>
            <ResourceMatch MatchId="urn:oasis:names:tc:xacml:2.0:function:ipAddress-regexp-match">
              <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">^(193)\.(140)\.(248)\.(162)\:(80)$</AttributeValue>
              <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
DataType="urn:oasis:names:tc:xacml:2.0:data-type:ipAddress" />
            </ResourceMatch>
          </Resource>
        </Resources>
      </Target>
      <Condition>
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
            <SubjectAttributeDesignator AttributeId="protocol"
DataType="http://www.w3.org/2001/XMLSchema#string" />
          </Apply>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">tcp</AttributeValue>
        </Apply>
      </Condition>
    </Rule>
...
 </Policy>
...
</PolicySet>
```

Fig. 3. Example XACML representation of a firewall policy rule

B. Representation of Topology

As explained above, three types of firewalls exist in this architecture: domain firewall which resides at the corporate premise, subnet firewall which protects its subnet within the domain and personal firewall which is controlled by the end-user. A subnet firewall is always between domain firewall and personal firewall. There might be concatenated subnet firewalls in the network topology, although there is only one domain firewall as the root of this tree based network topology and personal firewalls are always at the leaves of the topology tree. To represent this tree based topology we developed a DTD (Document Type Definition) for the topology XML file, which is shown in Fig. 4.

A topology element consists of one or more subnet elements and a subnet element consists of one or more firewall elements. A subnet element has only a description attribute, whereas a firewall element has two attributes: Firewall Type (the type denotes whether it is a domain, subnet, or personal firewall)



```
<?XML version="1.0"?>
<!DOCTYPE Topology [
        <!ELEMENT Topology (Subnet)*>
        <!ELEMENT Subnet (Firewall)*>
        <!ATTLIST Subnet Description CDATA #REQUIRED >
        <!ELEMENT Firewall>
        <!ATTLIST Firewall Type (DFW | SFW | PFW) #REQUIRED IP-Address CDATA #REQUIRED >
]>
```

Fig. 4. A DTD for topology representation

and IP-Address (the IP address of the firewall).

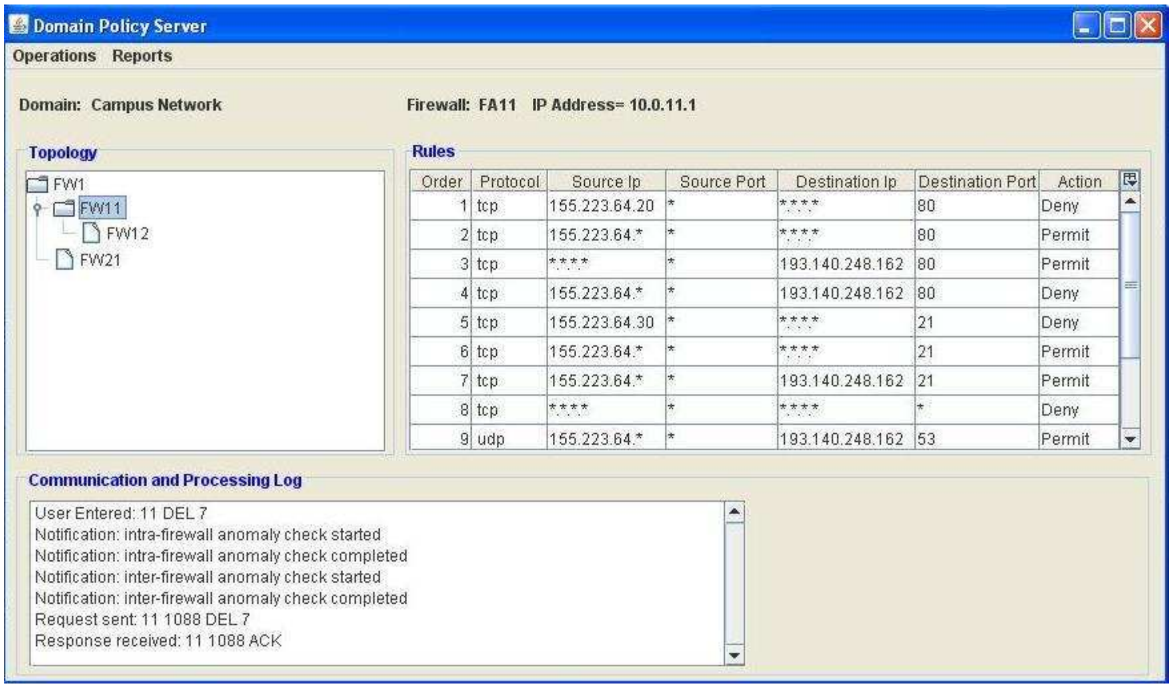
C. Representation of Configuration Messages

Firewall configuration messages are in the form of a request and a response, regardless of the party starting the communication. Their information content is as follows:

Request : $FA\#\ SEQ\#\ command\ order|ordered-rule$

Response : $FA\#\ SEQ\#\ ACK|NACK\ \varnothing|ordered-rule|exception$

Since there is only one DPS, there is no need to identify that party in the configuration messages. Only the firewall



Domain: Campus Network

Firewall: FA11 IP Address= 10.0.11.1

Order	Protocol	Source Ip	Source Port	Destination Ip	Destination Port	Action
1	tcp	155.223.64.20	*	*.*.*.*	80	Deny
2	tcp	155.223.64.*	*	*.*.*.*	80	Permit
3	tcp	*.*.*.*	*	193.140.248.162	80	Permit
4	tcp	155.223.64.*	*	193.140.248.162	80	Deny
5	tcp	155.223.64.30	*	*.*.*.*	21	Deny
6	tcp	155.223.64.*	*	*.*.*.*	21	Permit
7	tcp	155.223.64.*	*	193.140.248.162	21	Permit
8	tcp	*.*.*.*	*	*.*.*.*	*	Deny
9	udp	155.223.64.*	*	193.140.248.162	53	Permit

```
User Entered: 11 DEL 7
Notification: intra-firewall anomaly check started
Notification: intra-firewall anomaly check completed
Notification: inter-firewall anomaly check started
Notification: inter-firewall anomaly check completed
Request sent: 11 1088 DEL 7
Response received: 11 1088 ACK
```

Fig. 5. Graphical user interface for firewall configuration management

agent number denoted by FA# should be in the messages. Sequence numbers denoted by SEQ# are randomly generated and included into the messages. Authentication of configuration messages is critical for such an architecture and it is planned as a future work.

The commands inserted into a request message are listed in Table 1 along with their parameters. For ADD command, the ordered-rule should exist in the request message whereas for DEL and GET commands existence of the order is sufficient. Each request is replied with a response message, which is actually a positive or negative acknowledge message. If the response is to the ADD or DEL commands and it contains a positive acknowledge, nothing else is sent to the requester. Only a GET command is replied with a positive acknowledge and an ordered-rule. All negative acknowledge responses contain an exception string, which indicates the cause of the exception. Currently the only exception generated by the implementation is ”anomaly detected” exception coming from DPS. If such is the case, it means that the request is not accepted and the FA should not execute the command on the Iptables firewall.

V. Implementation

We developed a prototype implementation of the proposed architecture. The main component of our architecture, the Domain Policy Server, is implemented in Java 1.5 to be platform independent. For the XACML representation and manipulation of firewall policies we used libraries of SUN's XACML implementation [13]. XACML reporting of firewall policies is achieved through the use of UMU's XACML graphical user interface [9]. Graphical user interface of our prototype implementation is presented in Fig. 5.

The program window has four main components; namely information panel at the top, topology panel at the left, rules panel at the right, and communication and processing log panel at the bottom. The information panel presents information about the domain and selected firewall. The selection operation is performed on the topology panel. The topology information is imported from topology.xml file stored in the policy repository once the program is started. After a firewall is selected, its rules are listed in ordered-rule tuple format on the rules panel. Requests and responses from either party are printed on the communication and processing log panel. In Fig. 5, the logs of an administrator operation can be seen, where one would like to delete 7th rule of firewall (agent) numbered as 11. After entering this command from a separate window, the DPS performs necessary anomaly checks and if no anomaly is found then the request is sent to firewall 11 automatically. Firewall 11 replies with a positive acknowledge meaning that operation is successfully committed.

The firewall agent prototype is implemented in Phyton 2.4 on Linux machines. The prototype implementation assumes that all types of firewalls considered in our architecture are Iptables firewalls running on Linux machines. The FA has command console interface and commands entered are first sent to DPS through a socket connection, which is assumed to be always open. If DPS replies with positive acknowledge, the FA executes related Iptables command(s).

VI. Experiments and Evaluation

In order to obtain operational values for our prototype implementation of the proposed firewall configuration man-

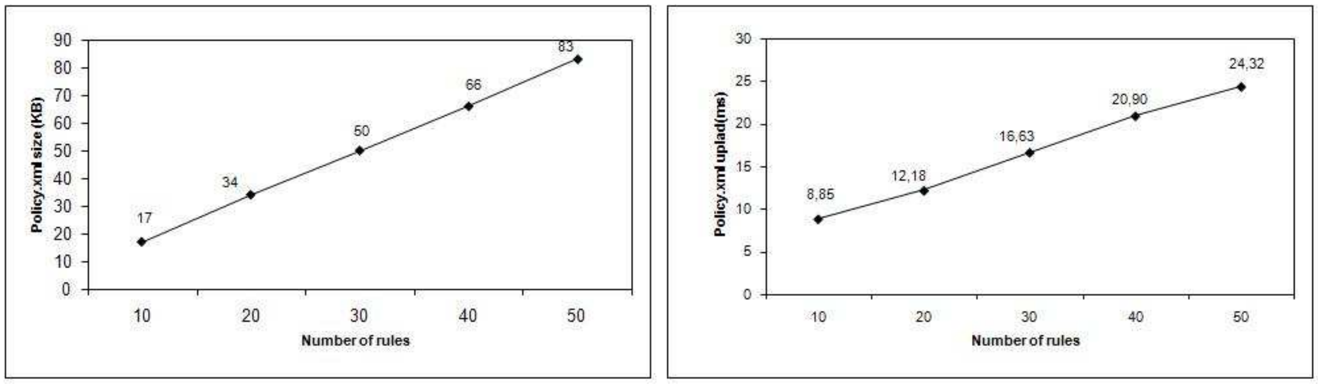


Fig. 6. Experimental values obtained for Policy.xml files (a. average file size) (b. average upload time)

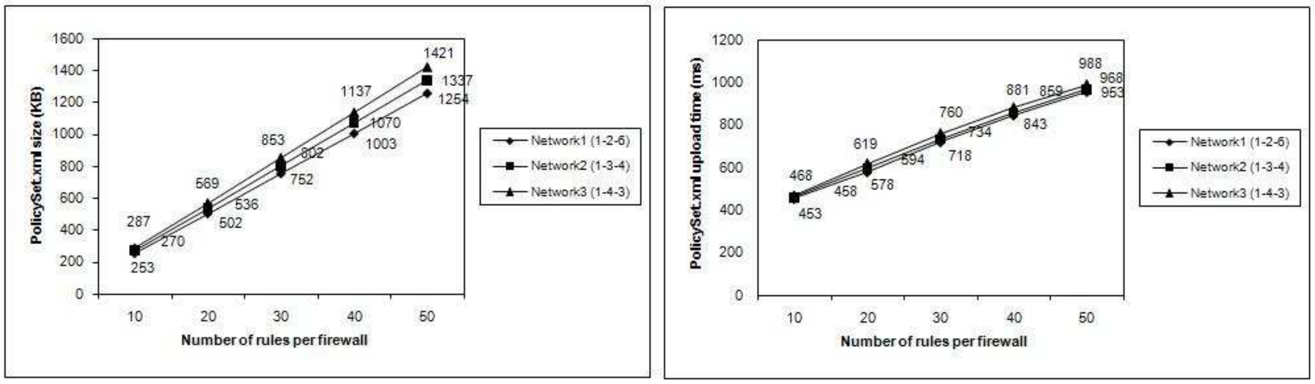


Fig. 7. Experimental values obtained for PolicySet.xml files (a. average file size) (b. average upload time)

agement architecture, a number of experiments are performed using different policies and network topologies. These experiments are performed on a Pentium IV-M 1.73 GHz. processor with 1.5 GByte of RAM, which is used as the domain policy server.

In the first experiment, randomly generated various sizes of firewall rules (10-50 rules) are represented in XACML for a single firewall and stored in a Policy.xml file. The average file sizes are shown in Fig. 6a and Fig. 6b presents average upload time of these files.

In the second experiment, instead of a single firewall we created network domains containing 15, 16, and 17 firewalls with the following topologies respectively; (1-2-6), (1-3-4), and (1-4-3). For instance, a (1-3-4) topology means one domain, three subnet, and twelve personal in total 16 firewalls. Same randomly generated firewall rules (10-50 rules) are used for the second experiment and all the policies of firewalls that exist in the domain with respect to topology are stored in a PolicySet.xml file. The average file sizes are shown in Fig. 7a and Fig. 7b presents average upload time of these files.

Same randomly generated firewall rules (10-50 rules) are used for the third experiment as well. This time, Policy Anomaly Checker (PAC) is used to execute intra-firewall anomaly discovery algorithm [2] on each set of generated firewall rules. In each case, processing time is measured. The results are shown in Fig. 8a. PAC is then used to execute inter-firewall anomaly discovery algorithm [2] for the network topologies defined in the second experiment. For each network topology, the processing time required to produce the final policy anomaly result is measured. The results are shown in Fig. 8b. PAC processing time increases exponentially with respect to number of rules per firewall. Moreover, PAC processing time increases with respect to number of subnet firewalls.

VII. Conclusion

This paper describes an architecture for managing configurations of firewalls distributed over a network domain. The main components and their operations are described in addition to XML representation of information used within the proposed architecture. The most important contribution of the paper is to use XACML for the representation of distributed firewall policies. With the prototype implementation of the proposed architecture it was possible to measure average size of XACML based policy files and their upload time. Moreover, we generated XACML based PolicySet.xml files for three different network topologies and measured policy anomaly checker processing time for different number of firewall rules.

The proposed architecture has a lot of features that requires further development. One important future work is to include

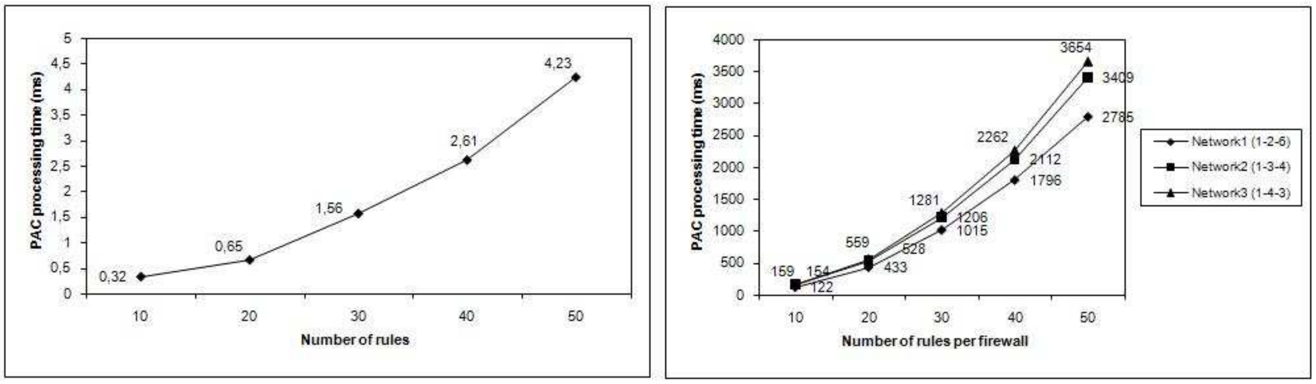


Fig. 8. Average PAC processing time (a. for intra-firewall anomalies) (b. for inter-firewall anomalies)

a secure policy exchange protocol that provides authentication, data integrity, confidentiality and access control. We need to extend our architecture so that management of a number of domains distributed over globe is possible. In that case, at the top hierarchy there will be a corporate policy server which manages domain policy servers. Furthermore, firewall agents can be designed to be more intelligent so that they observe their network and send information, like abnormal traffic, to the domain policy server.

References

[1] E. S. Al-Shaer and H.H. Hamed, *Design and Implementation of Firewall Policy Advisor Tools*, DePaul CTI Technical Report, CTI-TR-02-006, 2002.

[2] E. S. Al-Shaer and H.H. Hamed, *Discovery of Policy Anomalies in Distributed Firewalls*. In Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies, Hong Kong, China, 2004.

[3] C. Ardagna and S. C. Vimercati, *A Comparison of Modeling Strategies in Defining XML-based Access Control Languages*, Comput. Syst. Sci. Eng., 19(3), 2004.

[4] J. Burns, A. Cheng, P. Gurung, S. Rajagopalan, P. Rao, D. Rosenbluth, A. V. Surendran and D. M. Jr. Martin, *Proceedings of DARPA Information Survivability Conference & Exposition II*, Page(s):12 - 26, vol.2, 2001.

[5] F. Cetin, O. Yarimtepe and T. Tuglular, *Guvenlik Duvarlari Icin Politika Anomali Belirleme*. Akademik Bilisim 2008 Konferansi, Canakkale, Turkey, 2008.

[6] V. Cridlig, R. State and O. Festor, *An Integrated Security Framework for XML based Management*. 9th IFIP/IEEE International Symposium on Integrated Network Management, Nice, France, 2005.

[7] E. Jamhour, *Distributed security management using LDAP directories*. Proceedings of the XXI Internatinal Conference of the Chilean Computer Science Society, Punta Arenas, Chile, 2001.

[8] R. Kay, *Analysis: XACML*. Retrieved 20 January, 2008, from http://www.computerworld.com.au/index.php?id=997396262&fp=16&fpid=0, 2003.

[9] *UMU-XACML-Editor*, Retrieved 20 January, 2008, from http://xacml.dif.um.es, 2007.

[10] Matthew Burnside, Angelos D. Keromytis. *Arachne*, Integrated Enterprise Security Management, Information Assurance and Security Workshop. IAW '07. IEEE SMC, West Point, NY, USA, 2007.

[11] R. Mohan, T.E. Levin and C.E. Irvine, *An Editor for Adaptive XML-Based Policy Management of Ipsec*, Proceedings of the 19th Annual Computer Security Applications Conference, Las Vegas, Nevada, 2003.

[12] Nathan N. Vuong, Geoffrey S. Smith, Yi Deng. *Managing Security Policies in a Distributed Environment Using eXtensible Markup Language XML*, Proceedings of ACM Symposium on Applied Computing, Las Vegas, United States, Pages: 405 - 411, 2001.

[13] SUN. *Sun's XACML Implementation: Programmer's Guide for Version 1.2*, Retrieved 20 January, 2008, from http://sunxacml.sourceforge.net/guide.html, 2004.

[14] H. Yonggang, *Context Aware Security Policy Enforcement: CASPEr*, Unpublished Master's Thesis, Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, Holland, 2005.

[15] T. Yoshioka, T. Igakura and T. Tonouchi, *Policy-based Automatic Configuration of Network Elements in Separate Segments*. In Proceedings of (APNOMS 2003) Asia-Pacific Network Operations and Management Symposium, 2003.