An XML-based Approach to Combine Firewalls and Web Services Security Specifications

Marco Cremonini
cremonini@@dti.unimi.it

Ernesto Damiani
damiani@dti.unimi.it

Sabrina De Capitani di Vimercati
decapita@dti.unimi.it

Pierangela Samarati
samarati@dti.unimi.it

Dipartimento di Tecnologie dell'Informazione
Università di Milano
26013 Crema - Italy

ABSTRACT

The Web Services Architecture (WSA) defines a comprehensive model for service-oriented interactions among endpoints over a private network or the Internet. Since the many opportunities for better interacting services and the provision of richer functionality, crossing the boundary of organizations many standard proposals addressing different aspects of such interaction model are appearing. In this paper, we analyze the security requirements of the WSA and observe that the security model currently developed is not sufficient. In particular, we claim that many aspects related to network security and the integration of firewalls into the WSA have been underestimated. We show with different examples the usefulness of a semantics-aware firewall operating both at SOAP level and at lower network-based layers. We analyze, under this perspective, the impact on security that recently proposed stateful SOAP-based protocols could have, and describe how asynchronous protocols could pose high security risks on both service providers and service requesters. This drives us to the conclusion that, if security is an enabling factor for the success of Web service technologies, then perimetral security and firewall technology should be both fully supported into the WSA and improved to satisfy the requirements of the service-oriented interaction.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; C.2.6 [**Computer Communication Networks**]: Internetworking; D.4.6 [**Operating Systems**]: Security and Protection

General Terms

Security

Keywords

Web Services, SOAP, firewall, network security, service security

1. INTRODUCTION

The Web Services Architecture (WSA) [3] identifies the architecture's essential functional blocks and their interrelationships, and endorses the fundamental technologies (SOAP [15] and WSDL [5]) required to claim WSA-conformance. A *Web service*, as defined in the WSA, is a software system identified by a URI, whose public interfaces are defined and described using XML. Interactions with other systems are performed according to the Web service definition and using XML messages conveyed by Internet protocols. The WSA assumes the existence of a global network of Web services and specifies how this service network could interoperate in a reliable, manageable and secure manner, how it scales up, and how it could be extended and integrated with the World Wide Web.

The WSA represents a particular instance of the more general class of *Service-Oriented Architectures* (SOA). A SOA is a distributed system in which the interacting entities are services. A service is a software component providing for some well-defined functions that could be invoked through a network-addressable interface by means of standard protocols and data formats. Given this, one basic difference between the WSA and the WWW is that in the former messages are exchanged between Web services for the purpose of requesting and provisioning services [3], while in the latter messages are targeted on exchanging information.

As far as Internet protocols are concerned, Web services growing success hints to a scenario where HTTP, with its secure variant HTTPS, is the only general-purpose application protocol used for a variety of services.

Security is a core feature virtually spanning at all levels of the WSA, from HTTP communication to top-most features like discovery, aggregation, and choreography. Up to now, proposals are under development by both industry and academia [7, 8, 9, 10] and standards are emerging (e.g., WS-Security [1], XACML [12], and SAML [14]).

However, despite all efforts toward securing Web services, security concerns are still reported to be one of the strongest reason for delaying the adoption of this technology among companies and negative comments frequently comes from the security community pointing out the adverse effects that Web services will have upon the protection of companies' IT assets.

Our opinion and the rationale of this paper is that to explain these conflicts it is not sufficient to recall the generic tension between security and functionality, because in this case we have on the one side the WSA integrated with excellent standards for securing services and messages, and on the other side the network security community strongly argumenting against it. To tackle the problem of building a secure WSA then, we should analyze the WSA augmented with service-oriented security measures on the one side and the network security on the other side, which currently results in a clash of two opposite visions. We will then discuss the role of firewalls in the WSA, and how firewalls could evolve to meet the characteristics of the WSA as well as at which degree the WSA is now supporting the requirements of an effective perimetral security.

Our underlying assumptions are that a weakened perimetral security will result in a net loss of security for the whole WSA and that an integration of the WSA with the network security is both worthwhile and possible. However, the debate on these issues is still at a preliminary stage: while enterprises have quickly come to realize the importance of securing XML, they are only beginning to understand the many operating issues involved in implementing XML security at the single service and at the perimetral level and the corresponding costs.

The paper is structured as follows: Section 2 discusses the different visions about security measures currently developed for the WSA and for network security. Section 3 presents our proposal for an integrated model combining Web Services and firewalls security. Section 4 analyzes stateful SOAP protocols and discusses how such protocols could be managed. In particular, for the asynchronous mode, the serious security risks that these protocols might pose are analyzed. Section 5 analyzes some traditional security issues applied to the WSA showing possible solutions to mitigate the adverse effects. Finally, Section 6 draws our conclusions and sketches our plans for future works.

2. SERVICE-ORIENTED SECURITY VS. NETWORK SECURITY

One explicitly mentioned design benefit of WSA is that tunneling remote procedure calls and corresponding responses through general purpose HTTP connections,[1] instead of supporting them with specific application protocols, greatly improves the interoperability of the system. This has driven the development of a specific security model that we call *Service-Oriented Security* model. A Service-Oriented Security model fits well into a Service-Oriented Architecture like WSA, where the global system is seen as a network of endpoints requesting or providing services each one according to its own definition. Within this model, the *security features* are focused on: *(i)* protecting messages in transit on network links between endpoints and *(ii)* regulating the interaction with endpoints according to their definition. The *security technologies* provided in the WSA are perfectly coherent with this model: *(i)* message delivery through the public network is accomplished by signing and encrypting messages to guarantee their integrity and confidentiality [1], and by sequencing messages using message IDs to avoid replay attacks [2, 4]; *(ii)* the access to endpoints' services is based on authorization engines that enforce access control policies.

In the WSA the presence of *intermediaries* between each pair of communicating endpoints is considered for reasons like routing, reliability, or security. In particular, when the reason is security, an intermediary can be a trusted third party for handling policies that cannot be disclosed, an authorization engine that checks security assertions (e.g., credentials), or a *firewall*. This latter case is peculiar since it is the only one for which specifications and models lack completely in the WSA.

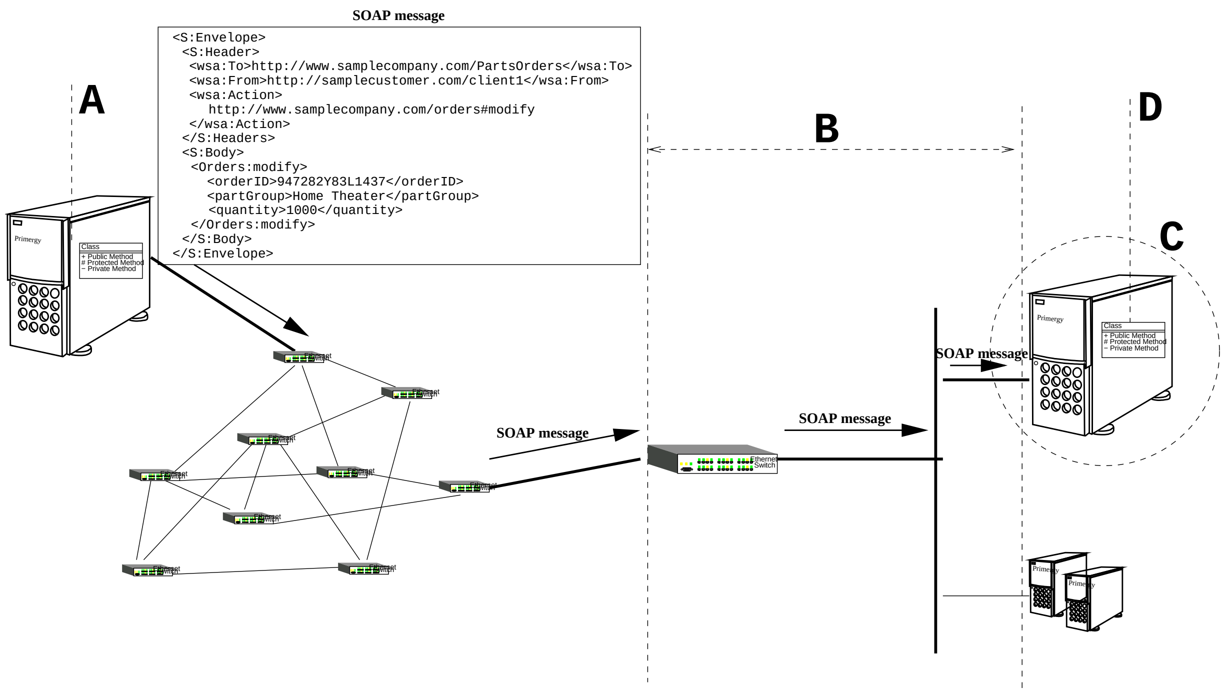
The reason for this anomaly, in our opinion, is that firewalling techniques do not fit well into the WSA and clash with the Service-Oriented Security model. This different and conflicting vision is well exemplified by the so-called *firewall traversal problem* that shows how, given the general problem of securing remote interactions carried by SOAP messages, the weakness of firewalls in the Network Security can be seen as a benefit in the Service-Oriented Security model. WSA assumes that it is beneficial for the overall interoperability that traditional firewalls were unable to handle tunneled protocols like SOAP because this would permit endpoints to directly communicate. The Network Security area, instead, regards Web services firewall traversal capabilities as a severe security risk because it encourages companies to selectively expose to the Internet pieces of their enterprise applications and networked hosts. Accordingly, it strongly discourages to deal with Web services security by access control alone, because a number of security and operating issues are more effectively solved at perimetral level. From the Network Security perspective, endpoints offering services should be limited and made unreachable from the Internet if they do not satisfy a corporate-wide, network-oriented perimetral policy.

To better understand why firewalls seem to conflict with the Service-Oriented Security model, the notion of resource in both models must be considered in more detail.

2.1 Resources in the Service-Oriented Security model and Network Security

The Service-Oriented Security model and Network Security protect resources that must be univocally identified (e.g., by means of URIs): in one case resources are services, in the other resources are everything reached or traversed by network packets. The two sets are not equivalent and this difference makes the two models conflicting. Let us introduce an example that will be used to clarify this fundamental difference. Figure 1 illustrates a scenario where the service requester `A` sends a SOAP message to the service provider `D`. The SOAP message includes the `To`, `From`, and `Action` elements defined by WS-Addressing [4], which represent, respectively, the URI of the intended receiver of the

[1]The WSA, as well as SOAP, is not protocol specific, although the bindings with HTTP is highlighted. Here we emphasize this binding assuming that it will be the one adopted in the majority of cases.



```
<S:Envelope>
 <S:Header>
  <wsa:To>http://www.samplecompany.com/PartsOrders</wsa:To>
  <wsa:From>http://samplecustomer.com/client1</wsa:From>
  <wsa:Action>
    http://www.samplecompany.com/orders#modify
  </wsa:Action>
 </S:Headers>
 <S:Body>
  <Orders:modify>
    <orderID>947282Y83L1437</orderID>
    <partGroup>Home Theater</partGroup>
    <quantity>1000</quantity>
  </Orders:modify>
 </S:Body>
</S:Envelope>
```

Figure 1: Interaction between two endpoints.

message, the URI of the intended sender, and the URI that identifies the semantics implied by the message.[2]

In particular, in this example the service requester `http://samplecustomer.com/client1` sends a SOAP message to the service provider `http://www.samplecompany.com/PartsOrders` to perform action `http://www.samplecompany.com/orders#modify` that consists in modifying an order. The `Body` includes the specific order, the product type, and the quantity.

Service-Oriented Security model. WS-Security technologies permit to sign and encrypt different parts of the messages. Then, regulating the service invocation requires authentication by the requester. This, for example, can be accomplished by means of SAML authentication methods [14], which can use passwords, Kerberos tickets, X.509 or PGP public keys among others. In addition, a *service-oriented security policy* could be enforced. A security policy regulates the way a requester can use the functionalities of the provided service, *within the domain of all legal functionalities* (i.e., defined by the service description). For instance, with respect to our example, a policy could say that *the number of items of type Home Theater that a requester can order must be less than 1000.* Service-oriented access control policies could be made complex as needed, according to the nature and number of services, as well as defined based on classifications of requesters and providers [10].

To summarize, the Service-Oriented Security model makes it possible to:

[2]The example is intended as a general case, although the three elements considered are specific of WS-Addressing. Other specifications (e.g., WS-Routing or WS-Reliability) prescribe to carry as message headers similar information regarding the endpoints and the purpose for which a message is intended for.

- guarantee the message integrity and confidentiality;
- provide authentication features between the endpoints of the communication;
- protect (groups of) services provided by the ultimate endpoint of a SOAP communication. Policy enforcement can then be co-located with the service provider or done by an intermediate node.

Network Security model. Saying that a service-oriented access control policy regulates only those service accesses within the domain of all legal functionalities implies that there could be illegal functionalities, or ways to induce side-effects and unforeseen behaviors to provided services without breaking the access control policy or bypassing its enforcement. For instance, denial of service, malformed packets/messages, buffer overflows, SQL injection techniques, trojaned services are examples of Web application vulnerabilities that are very likely to apply also to a WSA. Therefore, there are resources to protect other than those considered by the Service-Oriented Security model. This drives us to analyze the application of the Network Security model to the example in Figure 1. With the Service-Oriented Security model, we are guaranted that SOAP messages travel securely from the service requester `A` to the service provider `D` and safely interact with `D`'s functionality. Instead, with Network Security, resources reached or traversed by network packets are: *the service provider's organization own network* `B`; and *the service provider's hosting environment* `C`.

The *organization own network* is a private asset worth to be protected from undesired network traffic that could cause a reduction in the quality of service or even connection failures. The private network can be seen as a unique resource or as composed by several subnets, each one representing a

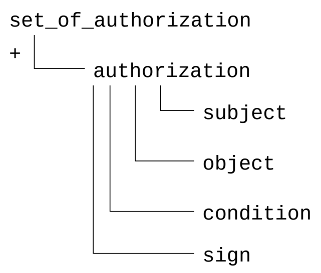


Figure 2: Structure of the authorization rules.

resource to be used according to some security policies. The Service-Oriented Security model does not take into account these resources, although it could, since the private network and its subnets are univocally identified with URIs and provide services to remote requesters (i.e., connection services are provided by network devices such as routers, gateways, or switches).

With *service provider's hosting environment* we intend everything supporting and enabling the provision of a certain Web service. With this definition we include, for example, the operating system, the application platforms (e.g., web server, DBMS, legacy systems) and the actual implementation and run-time environment of services.

3. THE INTEGRATED MODEL

The Service-Oriented Security model and the Network Security model can then be integrated thus obtaining a complete secure infrastructure to XML Web services. In particular, by starting from the work described in [10], we propose a model whose authorization rules have the form illustrated in Figure 2. We use the proposal in [10] because it is simple. We note however that emerging standards (e.g., XACML [12]) could be used for this purpose.

Subject. The subject field defines the *parties* to which the authorization applies. Our model supports the specification of a subject by making a reference to any feature possible specified in a custom header included in each SOAP call. By exploiting the tree format of a SOAP message (typical of all XML-based documents), a subject feature can be referenced by means of a path expression [6]. While this is the more general and powerful mechanism that allows to capture all the information characterizing a subject, it strictly refers to the syntax of the elements in the SOAP message. Therefore, our model also provides a higher level support for the specification of subjects. More precisely, a customizable set of XML elements can be defined each of which corresponds to a subject feature. For instance, element `from` can be defined to refer to the URI, or, more generally, to refer to the identifier of the sender of a SOAP message.

Object. The object field defines the service(s) to which the authorization applies. Like for subjects, we can use both generic path expressions and a customizable set of elements. Also, services can be defined by specifying a boolean formula of features that can evaluate membership of the service in groups or values of any of its parameters.

Condition. The condition field defines additional restrictions that have to be satisfied by the subject in order to access the object. As we will see later on, the condition field can be used to define network-based restrictions useful for avoiding or limiting attacks such as denial-of-service, spoofing and so on.

Sign. The sign field states whether the authorization defines a permission (+) or a denial (−).

Figure 3 illustrates some simple examples of authorizations. The first authorization states that the number of items of type `Home Electronics` that a `Retailers` (we suppose that the SOAP header includes information about subject groups as defined in [10]) can order must be less than 1000.

The second authorization regulates SOAP messages that have both `PartOrders` as the service provider and the destination IP address included in the subnet `159.149.16` (this is not to be confused with NAT functionality, where the destination IP address is changed according to a private IP addressing space). The subject and condition components of the authorization are empty meaning that it applies to any subject without restrictions. A rule like this could be helpful to prevent the illicit installation of rogue service providers inside the network, to force Web service providers to be located on a specific subnet, or even to deny the routing of crafted messages made for scanning purposes or other unauthorized actions (e.g., when the destination IP address is that of a host not providing for the specific Web service). This rule could be enforced in at least two ways: *(i)* SOAP messages explicitly carry an header block for the IP address of the destination, possibly inserted by a SOAP intermediate node, before being filtered by a semantics-aware component operating on SOAP headers only; or *(ii)* when the rule is enforced, it must be recognized that the condition upon the destination subnet is referred to an IP header, not a SOAP one, therefore a packet filtering component must be activated and its result combined with the result of the condition upon the `To` header. In any case, the enforcement of such a authorization cannot be done locally to the service provider's hosting environment, since the rule is actually defining the access to a composite service formed by both a Web service provider and a networking service.

Finally, the third authorization states that whether the request is directed to the service provider `PartOrders`, then the `Action` element must necessarily specify an *http* URL (the `www.samplecompany.com/` domain) and a Web method in the `orders` namespace. The benefits of this authorization is to prevent traffic made up by crafted SOAP messages carrying improper Web methods for a certain service provider. This could be made for scanning purposes, for example, trying to exploit information carried by the SOAP fault responses. By filtering these illegal messages at the corporate perimeter, we preserve, at least, the service provider hosting environment from wasting computational resources for processing them and for producing a fault message response. It is also important to note that out-of-specs messages often pose severe risks of malfunctioning due to their possible incorrect handling by the receiver.

For instance, the following Figure 4 is an example of rule that should check that whether the request is directed to the service provider `PartOrders`, then the *Ac-*

```
<subject>
   /S:Envelope/[S:Header/xsd:requesterGroup = "Retailer"]
</subject>

<object>
   /S:Envelope/[S:Body/Orders:modify/quantity < 1000 AND
                 S:Header/Orders:PartGroup = "Home Electronics"]
</object>
<condition/>
<sign value = "+"/>
```
–1–

```
<subject/>

<object>
   <IPheader><destinationIPaddr>159.149.16.*</destinationIPaddr></IPheader>
   <AND/>
   /S:Envelope/[S:Header/wsa:To = "http://www.samplecompany.com/PartOrders]
</object>
<condition/>
<sign value = "+"/>
```
–2–

```
<subject/>

<object>
   /S:Envelope/S:Header/[wsa:To = "http://www.samplecompany.com/PartOrders AND
                        wsa:Action = "http://www.samplecompany.com/orders#*]
</object>
<condition/>
<sign value = "+"/>
```
–3–

Figure 3: Examples of authorizations.

tion header must necessarily specify both an *http* URL, the `www.samplecompany.com/` domain and a Web method in the `Orders` namespace.

4. THREATS FROM STATEFUL SOAP PROTOCOLS

A wide area where security problems may arise and may need the integration between the service-oriented and the network approaches is represented by *stateful SOAP protocols*. Consider, for example, the *acknowledgment protocols* introduced for reliability purposes by WS-ReliableMessaging [2] and WS-Reliability [11]. According to these two sets of specifications, each SOAP message sent by a certain service to another (e.g., from a service requester to a service provider and vice versa) is acknowledged with an *ACK* message.

As an example, consider the messages in Figure 5.

In this case, reliable messages carry an header block `Sequence` composed by an `Identifier` of the message exchange and a `MessageNumber` for ordering messages within a certain sequence. The acknowledgment carries a `SequenceAcknowledgment` header block composed by the identifier of the sequence and a reference to the message number, one or more, acknowledged.

The purpose of these specifications is to provide assurances upon the message delivery like the transmission of all messages at most once, at least once, or exactly once (e.g., the *DeliveryAssurance* header can be used to specify all these modes). They even allow for ordering messages and possibly require re-transmission of lost messages.

Protocols governing a message exchange could be subject of several security risks, like eavesdropping (possibly addressed by encrypting sensitive information), replay attacks (possibly addressed by authenticating senders and discarding duplicate messages) or adverse effects due to out-of-specs messages. These examples represent some instances of the general problem of designing SOAP protocols resistant to attacks, which is a wide while still unexplored area.

4.1 SOAP acknowledgment protocols

We consider two possible ACK protocols, respectively called *synchronous* and *asynchronous* acknowledgment [11]. In the synchronous ACK protocol (see Figure 6(a)), the SOAP protocol for reliable messages is tightly binded with the HTTP *request-response* message exchange: each SOAP message is delivered as an HTTP Request and is acknowledged with the corresponding HTTP Response carrying the SOAP-ACK in the HTTP body. Differently, in the asynchronous ACK protocol (see Figure 6(b)), the binding between the SOAP protocol and the HTTP pattern of message exchange is weaker, since SOAP messages and SOAP ACKs are delivered on different HTTP connections. The asynchronous mode is clearly the more flexible since, for example, it permits to implement the single ACK for more messages according to the `AcknowledgmentRange` element shown in Figure 5.

Security issues that both service requesters and service

```
<subject/>
<object>
   /S: Envelope/S: Header/[wsa: To = "http://www.samplecompany.com/PartOrders AND
                           wsa: Action = "http://www.samplecompany.com/orders#*]
</object>
<condition/>
```

Figure 4: Example of condition upon the *Action* header.

a)

```
<S:Header>
  ...
  <wsrm:Sequence>
     <wsu:Identifier> http://www.samplecompany.com/abc </wsu:Identifier>
     <wsrm:MessageNumber> 10 </wsrm:MessageNumber>
  </wsrm:Sequence>
 </S:Headers>
```

b)

```
<S:Header>
  <wsrm:SequenceAcknowledgment>
      <wsu:Identifier> http://www.samplecompany.com/abc </wsu:Identifier>
      <wsrm:AcknowledgmentRange Upper="10" Lower="10" />
  <wsrm:SequenceAcknowledgment>
</S:Headers>
```

Figure 5: Example of WS-ReliableMessaging headers within *(a)* a reliable message and *(b)* an *ACK* message.

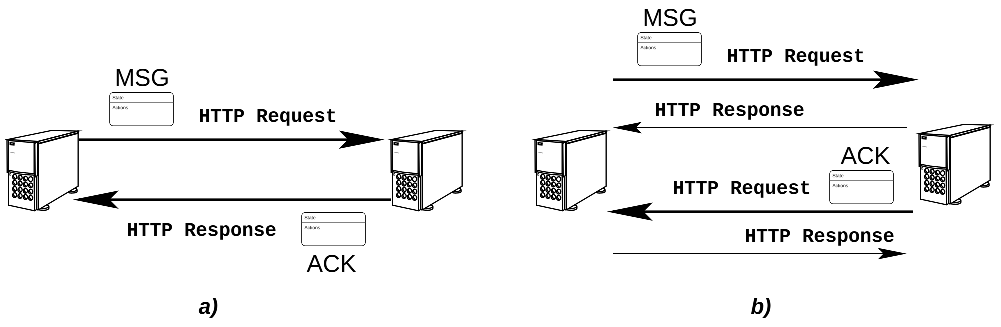


Figure 6: *a)* Synchronous and *b)* asynchronous acknowledgment.

providers should take into account can be summarized in two classes:

- *prevention from out-of-specs message exchange*: this is meant to deal with the situation of a malicious external party, either a requester or a provider, which does not follow the protocol message exchange pattern, for example, transmitting a SOAP-ACK not related with any previous message;

- *prevention from illegal communication channels*: this is meant to prevent that unauthorized communication channels were established, for example, by means of illegal service providers deployed inside a private network.

As a case study, we consider a simple security policy that is often enforced to prevent an illegal service provision: *for a given service, internal hosts should be able to act as service requesters toward external service providers and no external requests should be served by internal service providers.* This policy is often required by organizations willing to let their personnel access external services but worried about possible rogue servers installed inside the network.

An usual solution for TCP services is to require the setting of the TCP header's ACK flag for all incoming packets (i.e., optionally the same policy could be achieved by denying the setting of the TCP header's SYN flag alone for incoming network packets).

In our scenario with SOAP-ACK protocols, we may want to enforce the same security policy applied to Web services, instead of traditional TCP services. Let us consider whether the solution of ruling the value of the TCP header's ACK or SYN flags is effective when applied to the SOAP-ACK protocols.

Synchronous ACK. The policy could be enforced specifying rules at application and network level. Figure 7(a) illustrates two rules that control, respectively, the delivery of a SOAP reliable message and the reception of a SOAP ACK. Figure 7(b) instead illustrates a rule that drops all attempts to establish a TCP connection from an external client to an internal server. These three rules completely describe the required policy.

In more detail, the element `Sequence` of the first rule in Figure 7(a) identifies the SOAP reliable messages, while in the second rule the element `SequenceAcknowledgement` permits to capture SOAP-ACKs. The rule of Figure 7(b) (Linux `iptables` syntax [13] has been used) drops all incoming TCP packets (i.e., switch `-p tcp`) carrying the SYN flag alone set (i.e., switch `--syn`), resulting then in the denial of all attempts to connect to an internal server.

Asynchronous ACK. In this case regulating the message exchange according to the given policy is much more problematic. More precisely, it turns out that the filtering based on TCP flags is inapplicable if the delivery of SOAP messages and SOAP ACKs is carried out by distinct HTTP connections. Therefore, asynchronous protocols, which are the ones that support the development of rich functionality and complex message exchange patterns, pose a serious risk to Web service hosting environments because they may let all incoming connection attempts to be delivered unfiltered.

Let us come back to the solution presented for the synchronous case in Figure 7 and analyze it in the asynchronous scenario. It could be observed that: *i)* SOAP rules of Figure 7(a) are still applicable since they only specify that SOAP reliable message are let to go out and SOAP ACKs are let to come in; and *ii)* network rule, instead, is inapplicable since to let the SOAP ACKs in, upon a distinct HTTP connection, we need to enable an HTTP server to handle them.

Therefore, considering the interaction at TCP level, we have two endpoints acting both as clients and servers, where server ports are both standard HTTP ones (i.e., we are under the assumptions that the SOAP-HTTP binding is set and interoperability is assured by using standard HTTP connections). This means that at network level we cannot distinguish between a legal SOAP ACK message and an illegal attempt to establish a connection to the HTTP server. Hence, all incoming attempts to connect to the HTTP server of the requester could pass along unfiltered and actually succeed in establishing a TCP connection (i.e., the three-way handshake is completed). This holds in any case, even if content-based filtering mechanisms are in place since during the initial TCP handshake there are no data exchanged and only when data are transmitted, after the TCP session is established, content-based rules, like for example the second of Figure 7(a) could be enforced and the connection eventually dropped.

This is a first adverse effect of those protocols because the hosting environment where Web services are running is left unprotected from undesired network connections and this could permit attackers to try to exploit local vulnerabilities, misconfigurations and so on (note that HTTP servers are a traditional target of network attacks). Hence, a first observation is that, with asynchronous protocols, Web services endpoints, either provider or requester, are likely to suffer of many security problems of traditional Web application providers. Security of Web applications is a major issue today and comprehensive solutions lack. Architectural countermeasures, like the use of reverse proxies to screen the actual Web server, are then probably very useful even in a WSA.

A second adverse effect comes from the fact that, differently from a traditional Web architecture where Web clients are not forced to act as servers too, with these stateful asynchronous SOAP protocols even service requesters must provide for an active HTTP server accepting connections. If we consider that SOAP requesters could be deployed on a wide range of physical endpoints, from corporate hosts to personal computers or even PDAs, the security countermeasures for hardening the hosting environment may vary enormously. In particular, for personal computers or PDAs, it is unrealistic to think about rigorous hardening measures. This could potentially expose such endpoints to greater risks (e.g., of worm propagation) than with ordinary Web applications.

Possible solutions for handling SOAP-based stateful and asynchronous protocols could be achieved with the development of *stateful semantics-aware firewalls*, able to track the state of a SOAP communication and to fully inspect all message information at the different levels of encapsulation (e.g., Web Service, SOAP, XML, HTTP, TCP and IP). In this way, it would be possible to write rules that specify the type of messages in a very precise way and correlate attributes from all headers. For instance, we could think about conditions like:

- `<HTTPheader> POST </HTTPheader>` or `<HTTPheader> Response </HTTPheader>` to verify that the message complies with the HTTP binding;
- `<SOAPstatus> NEW | ESTABLISHED | RELATED |INVALID </SOAPstatus>` (state values are clearly taken from `iptables`), which should recognize if a SOAP message is the first of a new sequence, should correlate SOAP messages delivered by different HTTP connections according to their belonging to an existing message sequence (e.g., the reliable message and the ACK), or correlate fault messages.

Another measure for mitigating these security risks could be taken if the assumption of standard HTTP connections for interoperability and firewall traversal is re-considered and made more flexible. That is, if service provision is targeted to a close group of requesters, there is not such a need for standard HTTP connections that cannot be recognized as Web services message exchange from network filtering techniques. SOAP-HTTP message exchange could be binded to a non standard HTTP port, for example, making them recognizable with respect to conventional Web applications from the very first packet exchange. Service provider authentication and the possible use of technologies like Virtual Private Networks, is also extremely important since as we saw there could be greater security risks at the requester's side.

5. THREATS TO WEB SERVICE PROVISION

Service provision can potentially suffer of other threats caused by the reception of messages that cannot be handled properly or that can induce side-effects. In particular, two

a)

```
<subject>
   /S:Envelope/S:Header/[wsa:From: = "www.mycompany.com/*"
 </subject>
 <object>
   /S:Envelope/S:Header/[wsa:To: = "www.externalcompany.com/SampleService"]
 </object>
 <condition><messagetype>sequence</messagetype></condition>
<sign value ="+"/>

<subject>
   /S:Envelope/S:Header/[wsa:From: = "www.externalcompany.com/*"
</subject>
<object>
  /S:Envelope/[S:Header/wsa:To: = "www.mycompany.com/*"]
</object>
<condition><messagetype>sequenceacknowledgment</messagetype></condition>
<sign value ="+"/>
```

b)

```
iptables -A INPUT -p tcp --syn -j DROP
```

Figure 7: *a)* SOAP and *b)* network rules for the synchronous ACK protocol.

general security issues, denial-of-service and spoofing, are considered in relation to some specific features of the WSA.

Denial-of-Service. Assume that a malicious service requester crafts several sequences of SOAP messages all missing a message carrying a certain value in the `MessageNumber` element. Assume also that the messages are required to be delivered ordered and at least once. This case, if not handled properly, could result in a denial-of-service on the receiver side due, for example, to cache consumption if the service receiver keeps messages to be reordered in a cache before delivering them to an application handler. Time-out mechanisms could limit this case, although the delivery of multiple sequences with sufficiently high frequency could still result in a denial-of-service condition. Service provider hosting environments should be protected from this eventuality, since many services could be co-located on the same host, as well as a certain host could serve both requests from external and internal requesters.

Semantic-firewalls, aware of SOAP message protocols, could use *thresholds* mechanisms to mitigate this problem. This, together with a proper tuning of time-out mechanisms could be the first line of defense against denial-of-service attempts. Figure 8 shows an example of condition upon the frequency of invocation of a certain service provider and upon the number of simultaneous message sequences opened. Authorization `a`) is meant to limit to once per second the maximum rate at which requests coming from domain `http://www.samplecustomer.com/` could be addressed to the `PartOrders` service provider. Requests exceeding the threshold frequency must be dropped at the corporate perimeter without interfering with the service provider. However, it could be insufficient if the requester can use spoofed identities in the `From` header since the malicious requester could select different fake identities and overcome thresholds based on requester's identifier. General thresholds for service provider hosting environments could then be enforced, limiting the number of opened message sequences independently from the requester, as well as shorter time-outs could serve as possible countermeasures. Authorization `b`), in fact, is meant to limit to 5 the number of sequences of message exchange running simultaneously between any possible requester (i.e., `From: = *`) and the `PartOrders` service provider.

Spoofing. The possibility to craft SOAP messages by placing an identity in a `From` header not corresponding to the actual requester cannot be prevented unless strong authentication techniques are employed (e.g., by means of digital signatures). There are, however, many applications that cannot satisfy this requirement, for example when anonymous requesters are allowed, or could do that only in a weak form, for example when authentication is accomplished by checking the requester's domain or affiliation only instead of the personal identity. In other situations, the use of cryptographic techniques could be discouraged for performance reasons or excessive power consumption, for example whether the service request comes from a portable device.

Nevertheless, there is a case that is peculiar to SOAP messages with a `From` header block: requester identities are stored both in the `From` header block as a URI (e.g., as an email address or as an URL) and in the *source IP address* of the IP header as the network address of the source host. A potential enforcement upon the requester identity is to re-

a)

```
<subject>
   /S:Envelope/[S:Header/wsa:From: = "http://www.samplecustomer.com/"]
</subject>
<object>
   /S:Envelope/[S:Header/wsa:To: = "http://www.samplecompany.com/PartOrders"]
</object>
<condition>
   <Frequency> 1/sec </Frequency>
</condition>
```

b)

```
<subject>
   /S:Envelope/[S:Header/wsa:From: = *]
</subject>
<object>
   /S:Envelope/[S:Header/wsa:To: = "http://www.samplecompany.com/PartOrders"]
</object>
<condition>
   <SimultaneousSequences> 5 </SimultaneousSequences>
</condition>
```

Figure 8: Example of thresholds upon service invocation.

a)

```
<subject>
   <IPheader><sourceIPaddr/></IPheader> INCLUDED IN
                                        /S:Envelope/S:Header/wsa:From
</subject>
<object>
   /S:Envelope/S:Header/[wsa:To: = "http://www.samplecompany.com/PartOrders"]
</object>
<condition/>
<sign value="+"/>
```

b)

```
<subject>
   /S:Envelope/S:Header/wsa:From EQUAL TO
                 <IPheader><sourceIPaddr> 159.149.10.* </sourceIPaddr></IPheader>
</subject>
<object>
 …
</object>
<condition/>
<sign value="+"/>
```

Figure 9: Example of conditions to limit spoofing.

quire that the IP address resulting from de-referencing the logical name of the From header block must be related to the IP address stored in the IP header. This might result in an *EQUAL TO* relationship between the two (e.g., when we want the two are the same address) or in an *INCLUDED IN* relationship of the second within the first (e.g., when the From value results in a class of IP addresses containing the source IP address). It is worth to be noted that spoofing techniques that exploit a mismatch between the source IP address and the logical name in a From field are already widely used by worms that propagate themselves through email messages. Examples of these condition are shown in Figure 9. Authorization a) is meant to require that, for requests addressed to the PartOrders service provider, the source IP address is *included in* the IP address class corresponding to the URI of the From header. Authorization b), instead, is meant to require that the IP address corresponding to the URI of the From header is *equal to* the source IP address that belongs to the subnet 159.149.10.

6. CONCLUSIONS

The WSA is driving many expectations for a better interoperability of systems, either distributed within a corporate network or connected through the Internet and many interesting proposals aimed at introducing richer features have been produced. However, from the security viewpoint there seem to be an incomplete vision and actually some important while common issues related with network aspects seems to have been ignored.

Therefore, in this paper we have first analyzed the apparent clash between the Service-Oriented and the Network Security perspectives. In particular, we have discussed the requirements and presented possible design guidelines of semantics-aware firewalls fully integrated within the WSA. Throughout some examples we have also justified the utility of a perimetral management of the access control.

Discussing about some stateful SOAP-based protocols we have analyzed how they could pose serious security risks if not managed properly and in particular with mechanisms able to inspect the many encapsulated headers of a message and by having in place semantics-aware firewall able to track SOAP-level connections.

As we pointed out, security issues like the ones discussed in this paper are still largely unexplored. We plan to proceed with both our analysis of the many protocols and correlations among authorization attributes from different layers. The implementation of a prototype of a semantics-aware firewall is currently on-going.

7. ACKNOWLEDGMENTS

The work reported in this paper was partially supported by the Italian MURST within the KIWI and MAPS projects.

8. REFERENCES

[1] B. Atkinson and et al. *Web Services Security.* IBM Corp. and Verisign Inc., April 2002. http://www.ibm.com/developerworks/library/ws-secure/.

[2] R. Bilorusets and et al. *Web Services Reliable Messaging Protocol.* BEA Systems Inc., IBM Corp. and Microsoft Corp., March 2003. http://msdn.microsoft.com/ws/2003/03/ws-reliablemessaging/.

[3] D. Booth and et al. *Web Service Architecture.* World Wide Web Consortium (W3C), May 2003. http://www.w3.org/TR/ws-arch.

[4] A. Bosworth and et al. *Web Services Addressing.* BEA Systems Inc., IBM Corp., Microsoft Corp. and TIBCO Software Inc., March 2003. http://dev2dev.bea.com/technologies/webservices/ws-addressing.jsp.

[5] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. *Web Services Description Language (WSDL) 1.1.* World Wide Web Consortium (W3C), March 2001. http://www.w3.org/TR/wsdl.

[6] J. Clark and S. DeRose. *XML Path Language (XPath) Version 1.0.* World Wide Web Consortium (W3C), November 1999. http://www.w3.org/TR/xpath.

[7] M. Cremonini, E. Damiani, and P. Samarati. Semantics-aware perimeter protection. In *Proc. of the 17th IFIP Wg 11.3 Conference*, Colorado, USA, August 2003. to appear.

[8] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. Controlling access to XML documents. *IEEE Internet Computing*, 5(6):18–28, November/December 2001.

[9] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. A fine-grained access control system for XML documents. *ACM Transactions on Information and System Security*, 5(2):169–202, May 2002.

[10] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. Securing SOAP e-services. *International Journal of Information Security (IJIS)*, 1(2):100–115, February 2002.

[11] C. Evans and et al. *Web Services Reliability Ver1.0.* Fujitsu Ltd., Hitachi Ltd., NEC Corp., Oracle Corp., Sonic Software Corp. and Sun Microsystems Inc., January 2003.

[12] S. Godik and et al. *eXtensible Access Control Markup Language Version 1.0.* OASIS Open, March 2003. http://www.oasis-open.org/committees/xacml/repository.

[13] M. Josefsson and et al. The netfilter/iptables project, 2003. http://www.netfilter.org.

[14] E. Maler and et al. *Assertions and Protocols for the OASIS Security Assertion Markup Language V1.1.* OASIS Open, July 2003. http://www.oasis-open.org/committees/documents.php?wg_abbrev=security.

[15] N. Mitra. *SOAP Version 1.2 Part 0: Primer.* World Wide Web Consortium (W3C), May 2002. http://www.w3.org/TR/soap12-part0/.