Towards Secure SOAP Message Exchange in a SOA

Mohammad Ashiqur Rahaman, Andreas Schaad and Maarten Rits
SAP Research
805, Avenue du Docteur Maurice Donat
Font de l'Orme, 06250 MOUGINS
+33(0)4 92 28 62 00
{mohammad.ashiqur.rahaman, andreas.schaad, maarten.rits}@sap.com

Abstract

SOAP message exchange is one of the core services required for system integration in Service Oriented Architecture (SOA) environments. One key concern in a SOA is thus to provide Message Level Security (as opposed to point to point security). We observe that systems are communicating with each other in a SOA over SOAP messages, often without adequate protection against XML rewriting attacks.

We have already provided a solution to protect the integrity of SOAP messages in earlier work [1]. This solution was based on the usage of message structure information (`SOAP Account`) for preservation of message integrity. However, this earlier work did not discuss the issue of forging the `SOAP Account` itself. In this paper, we discuss the integrity feature of a `SOAP Account` within a more general context of the current web service security state of the art.

Categories and Subject Descriptors

D.2.11 [**Software Architectures]** D.4.6 [**Security and Protection**]

General Terms

Security, Design, Verification.

Keywords

SOA, SOAP Account, XML Rewriting Attack.

1. Introduction

A service oriented architecture (SOA) is a collection of loosely coupled services available in the World Wide Web [15]. Loose coupling means that the way a client (which can be another service) communicates with the service does not depend on the implementation of the service. The concept of a SOA is, however not new [14]. One of the first service-oriented architectures was the use of DCOM or Object Request Brokers (ORBs) based on the CORBA specification. Figure 1 shows a basic SOA. A service consumer sends a service request message to a service provider and the service provider returns a response message to the service consumer. Optionally, a SOA can also include a service that provides a directory or registry of services. A service consumer can discover services by examining the registry.

Although the concept of a SOA has thus been used for years, the evolution of web services-based SOAs is still a field of ongoing Research & Development in industry and academia. We observe that with the emergence of web service standards, the integration of systems in an A2A or B2B fashion has become more and more accelerated. Message exchange is one of the core services required for system integration in SOA environments. This message exchange is usually performed via the SOAP protocol. Since messages may carry vital business information, their integrity and confidentiality needs to be preserved and SOAP Message exchange in a meaningful and secured manner remains a challenging part of systems integration.

Since SOAP is based on XML, one particular exploit is that of a XML rewriting attack which is a general name for a distinct class of attacks based on the malicious interception, manipulation, and transmission of SOAP messages in a network of communication system. Using WS-Security [2], WS-Policy [3] and other standards correctly on SOAP we can avoid XML rewriting attacks [4]. However, in practice, incorrect usage and application of these standards by the human being is very likely and leads to significant vulnerabilities.

In earlier work [1] we have shown that the usage of SOAP message structure information, which we refer to as `SOAP Account`, can be an efficient technique to detect rewriting attacks. Although using `SOAP Account` [1] we can detect XML rewriting attacks very early in the validation process by a legitimate receiver of a SOAP message, a `SOAP Account` itself might be a target of attackers. Therefore this paper[1] aims at providing an analysis of the integrity of a `SOAP Account` itself.

SWS'06, November 3, 2006, Alexandria, Virginia, USA.

[1] The work of A. Schaad and M. Rahaman has been sponsored under the EU IST-2004-026650 project "R4eGov".

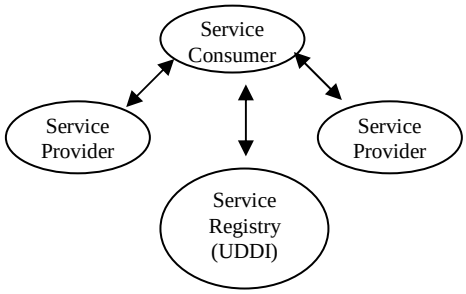

Figure 1: Service oriented Architecture

In this paper we describe web service security architectures in a simplified way using WS standards before addressing the issue of attacking a `SOAP Account` itself. We concentrate on message level security and thus show the necessity of message level security in web services. We use concrete scenarios showing how we achieve the integrity feature of a `SOAP Account` assuming the presence of a malicious attacker.

The paper is thus organized as follows. Section 2 discusses related work. Section 3 reviews related terminologies and techniques. Section 4 illustrates a state of the art web service based security architecture focusing on message flow and rewriting attacks with respect to a real-world business scenario. In Section 5 we then describe a scenario of a possible attack against a `SOAP Account` and reason about the `SOAP Account`'s integrity. Section 6 concludes this paper.

2. Related Work

Security in SOAs has been an active research field since the beginning of the SOA paradigm. This work is a continuation of our previous work [1] where we have presented and discussed an inline approach to include SOAP structure information in the SOAP message and to validate the information by the receiver of the message. In particular, we can attach SOAP Account information into the `<Security>` header in the WS-Security standard [2]. Essentially our `SOAP Account` has proposed a new SOAP header as any new standard in SOA does. However, we took performance issues into account as such an added SOAP header may introduce overhead in the processing of XML (such as XML canonicalization). We described a performance evaluation of the proposed technique to detect XML rewriting attack on SOAP messages which showed better performance when compared to standard policy based techniques [1].

In [13] the authors suggested to follow certain guidelines to integrate security aspects of web services throughout the development process of building web service based systems in service oriented architectures. They suggest an iterative and incremental model to incorporate web service security requirements, to design web service security architecture, and to select web service security standards for deployment. In addition, they describe a case study where they exercise the iterative and incremental model in the suggested way.

The SAMOA project [5] takes a formal approach to verify and validate web services specifications with rigorous techniques. SAMOA identifies common security vulnerabilities during security reviews of web services with policy-driven security [4] and proposes a tool named policy advisor to identify vulnerabilities automatically and to provide remedial advices. While their prior work [6] describes generating and analyzing web services security policies to detect XML rewriting attacks, this tool is able to bridge the gap between formal analysis and implementation quite efficiently. It also describes a formal semantics for WS-SecurityPolicy [7], and proposes an abstract link language [8] for specifying the security goals of web services and their clients.

3. Terminologies and Techniques

In this section we present the terminologies and techniques related to web services security that we later refer to in this paper and that have been widely deployed in industry. We also provide insights into the security context that is required in a SOA.

3.1. SOAP

SOAP [12] is a XML based messaging framework used to exchange encoded information (e.g. web service request and response) over a variety of protocols (e.g. HTTP, SMTP, MIME). It allows a program running in one system to call a program running in another system and it is independent of any programming model. SOAP provides an easy way to design protocols for communication between applications in an intranet or over the internet.

Since the emergence of SOAP, systems rely on the ability for message processing intermediaries to forward messages. Security information is contained within the SOAP message and/or SOAP message attachment, which allows security information to travel along with the message or attachment.

3.2. Point-to-Point Security vs. Message Level Security for SOAP Messages

Point-to-Point security context preserves the security context in between any two consecutive SOAP processing nodes as shown in Figure 2.

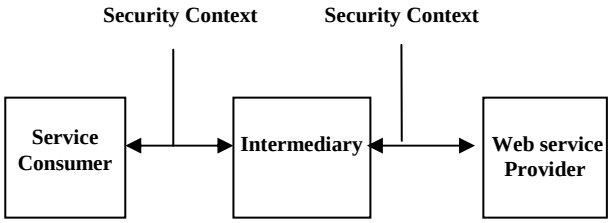

Figure 2: Point-to-point Configuration

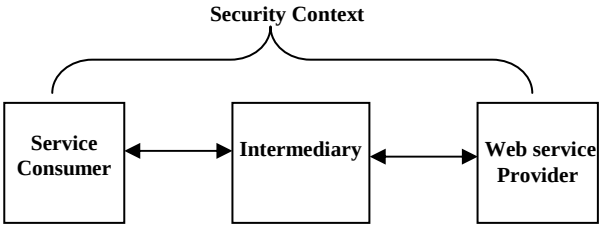

Figure 3: End-to-End Configuration

Transport level security (e.g. SSL, TLS) [16] supports Point-to-Point security context only (Figure 2) and does not handle End-to-End multi-hopped messaging security. So when a message is received and forwarded on by an intermediary (A SOAP processing node e.g. SAP XI or IBM Websphere) beyond the transport layer, both, the integrity of data and any security information that flows with it may be lost. This forces any upstream SOAP message processors to rely on the security evaluations made by previous intermediaries and to completely trust them with respect to their handling of the content of messages. Security is preserved only when data is on the wire, but not when off the wire (e.g. files, databases).

Using transport level security the current state of the art is invocation of HTTPS [17]. However, the communication is transient, Point-to-Point, and encrypted with known trusted parties which means that authentication of the parties and confidentiality of the data is guaranteed while data is in motion, but not while data resides within an intermediary. Web services can and do provide such features, but it is insufficient in several ways when transport level security is used:

- Transport Level Security is not granular enough because it encrypts everything.
- It is inflexible about routing because it is just Point-to-Point.
- Reduced auditing capabilities.
- Can not avoid repudiation because it is not signing the data.
- HTTP might not be the only transport that is used nowadays.

We need to adhere to more stringent security requirements for web services because:

- The point of interaction is more “over the internet” (as opposed to “within an intranet”).
- Interaction happens between partners with no previously established relationship.
- Program to program interaction (as opposed to human to program interaction).
- More dynamic interaction (as opposed to static interaction).
- Larger number of service providers and users.
- We need fine grained signature and encryption where element wise signing and encryption may be needed.

Message level security aka End-to-End security deals with and solves most issues of a transport level security scheme regarding its insufficiency, starting with maintaining a security context (Figure 3) which is valid End-to-End. The identity, integrity, and security of the message and the caller need to be preserved over multiple hops and more than one encryption key may be used along the route with trust domains being crossed.

4. Web Service Security

From a more general perspective, Web services describe the interaction of open WS* standards (e.g. SOAP, WSDL, UDDI), different implementation platforms (J2EE, .NET, ABAP), applications and devices. Active presence of such diverse systems makes it necessary to take an evolutionary approach that leverages the existing technologies to cope with the security concerns of a SOA. Web service specifications and techniques for secure SOAs have been evolving rapidly. SOAs provide loosely coupled applications to be composed and integrated from a set of internal and external services which are distributed over the internet.

In this section we present a simplified view of a web service security architecture considering the interplay of different Web Service standards and message flow when we deploy or implement the different WS* standards related to security in a simple sender and receiver scenario. We also provide a business scenario which is vulnerable to XML rewriting attacks.

4.1. WS Standards in Web Service Security Architecture

WS-Security [2], WS-Policy [3], WS-SecurePolicy [9] and other web service standards follow an evolutionary approach to address the End-to-End security context issue

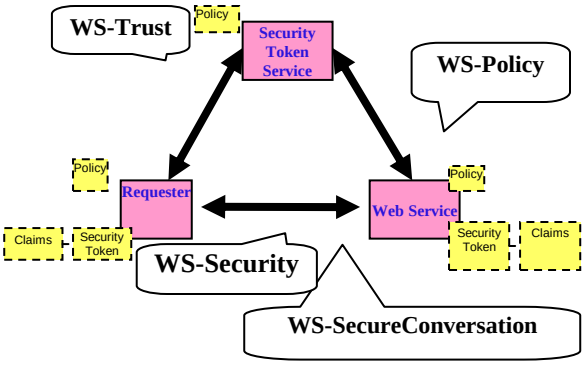

Figure 4: Simple Web Service Security Architecture

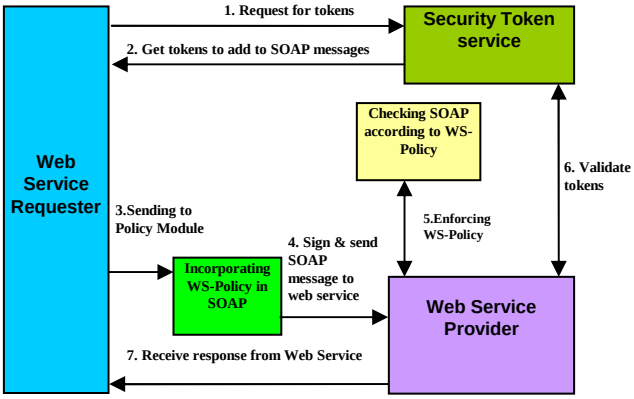

Figure 5: Typical message flow between web services using WS standards

in detail. Figure 4 shows a simple architecture of web service security considering different WS standards. Note that these mentioned standards play a central role in web service security architectures along with other standards.

WS-Security describes how to attach signature and encryption headers to SOAP messages as well as how to attach security tokens, including binary security tokens such as X.509 certificates and Kerberos tickets. WS-Security provides a framework to secure a SOAP message using existing techniques (e.g. encryption, signature).

WS-Policy and WS-SecurePolicy describe the capabilities and constraints of the security (and other business) policies on intermediaries and endpoints (e.g. required security tokens, supported encryption algorithms). For example, a service provider may only accept a X.509 security token which can be described using the declarative syntax of WS-Policy and WS-SecurePolicy.

As a SOA intends to provide the loose coupling of services the issue of having trust among the different entities (e.g. service provider, consumer, and intermediary) comes into play. WS-Trust [10] describes a framework for trust models that enables web services to securely interoperate. For example, a client can send only X.509 security tokens and the web service can accept only SAML security tokens. WS-Trust provides a protocol to get the SAML security token by presenting the X.509 security token. By doing so, WS-Trust resolves the token format mismatch; trust between client and web service can be established.

Using WS-Security independently for each message to secure a conversation is possible, but it is rather inefficient. WS-SecureConversation [11] describes how to manage and authenticate message exchanges between parties including security context exchange and establishing and deriving session keys.

Note that, though correct usage of all these standards can secure a SOAP message exchange in SOA, we observe some limitations to achieve the expected security [1]. We show an example of a possible attack in section 4.3.

4.2. Message Flow

On the sender side or Web Service Requester in Figure 5, at first the Requester will acquire the required security token from the Security Token Service and then the protocol stack generates SOAP envelopes that satisfy its policy. It adds integrity and confidentiality credentials under the `<Security>` header that is defined in WS-Security. The header block allows attaching security-related information targeted at a specific recipient in the form of a SOAP actor/role. This may be either the ultimate recipient of the message or an intermediary. Consequently, elements of this type may be present multiple times in a SOAP message. An active intermediary on the message path may add one or more new sub-elements to an existing header block if they are targeted for its SOAP node or it may add one or more new headers for additional targets.

Conversely, on the receiver side or Web Service provider, a SOAP envelope is accepted as valid and passed to the application if its policy is satisfied for this envelope. Normally, the sender policy should be at least as demanding as the receiver policy.

Figure 6: A SOAP request before an attack (Excerpt)

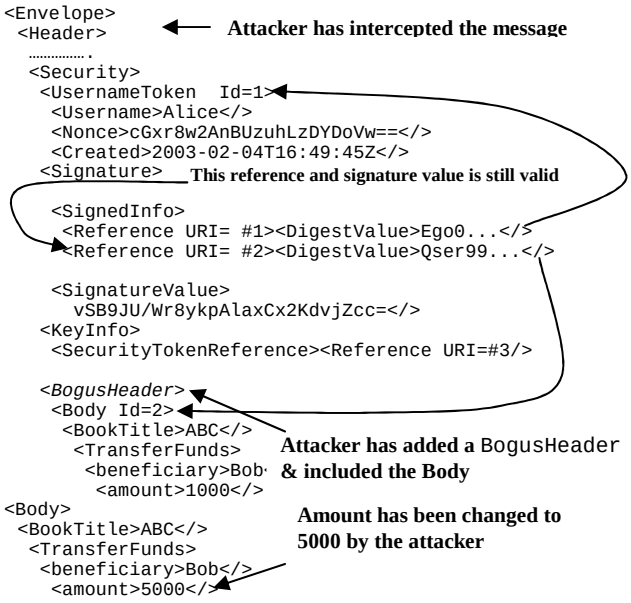

Figure 7: A SOAP request after attack (Excerpt)

4.3. Possible XML Rewriting Attacks in a Business Scenario

XML rewriting attack is a general name for a distinct class of attacks based on the malicious interception, manipulation, and transmission of SOAP messages in network of communication system. In this section we show a scenario of business processes that are vulnerable to such rewriting attacks.

Consider one service consumer of an online book shop service requests for some particular book and pays for it (Figure 6). Each successful request causes the consumer to pay. We assume that one SOAP node (Ultimate receiver) is supposed to process the SOAP header or Body. A customer, Alice, wants to transfer 1000 Euros from her account to the book shop's owner (Bob's) account (Figure 6) for a requested book. Some malicious attacker intercepts this message and updates it stating to transfer 5000 Euros instead of 1000 Euros (Figure 7). An attacker can now observe and manipulate the message on the SOAP path. He can introduce a new false header (e.g. `Bogus`) (Figure 7). Everything else, including the certificate and signature, remains same. The `<Bogus>` element and its contents are ignored by the recipient since this header is unknown, but the signature is still acceptable because the element at reference URI **"Id-2"** remains in the message and still has the same value. This may cause the consumer to pay several times for the same request and forces the service to do redundant work.

To detect the rewriting attack we add `SOAP Account` information in the SOAP message before sending it to the legitimate receiver. Figure 10 shows the SOAP message after adding `SOAP Account.` The rati-

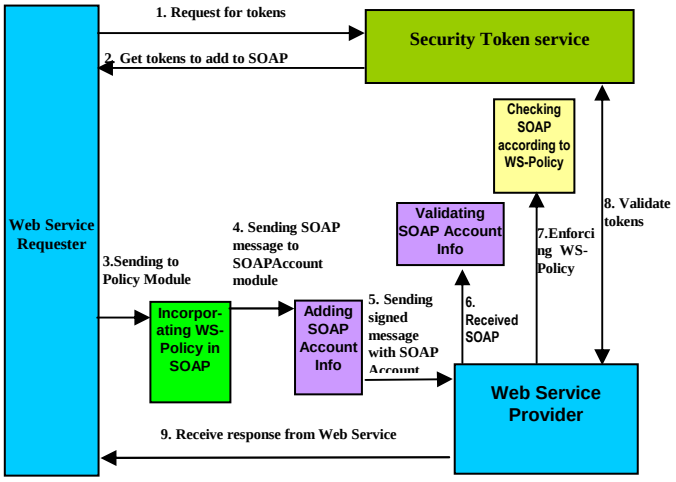

Figure 8: Message flow using new approach between web services

-onale behind using `SOAP Account` is described in detail in [1]. We have designed and implemented a module called `AddSOAPAccount` [1] to compute the `SOAP Account` information for every SOAP message that is exchanged in SOA environment. There is a corresponding `CheckSOAPAccount` module in every SOAP processing node which checks the safety of the received SOAP message as described in [1] and in the section 5.

Figure 8 shows the message flow when these modules are deployed in a SOA. The difference between Figure 8

and Figure 5 is the added SOAP Account module (AddSOAPAccount & CheckSOAPAccount) in a SOAP processing node (i.e. Sender, intermediary, receiver). Since new modules are added in every SOAP processing node the number of exchanged messages is increased by 2. A detailed performance analysis considering the added modules is given in [1].

4.4. SOAP Account

Our concept of a SOAP Account [1] refers to the general idea of keeping record of a SOAP message's structure of elements (e.g. Number of header elements, number of signed objects, and hierarchy information of the signed object).

Figure 9 shows the SOAP Account information that is used to detect the XML rewriting attacks in the scenarios of [1] and in this paper. As the main exploitation of the rewriting attacks was based on the structural syntax of a SOAP message, we focus on capturing the structure related information in a SOAP Account. We use the AddSOAPAccount [1] module to add this SOAP Account information into outgoing SOAP message.

5. Attacks against SOAP Account

A SOAP Account itself is vulnerable to XML rewriting attacks. Since the whole SOAP Account information is signed before sending it to the legitimate receiver any malicious attacker may try to forge it in the same way as in the scenarios described in the section 4.3. The usage of the CheckSOAPAccount [1] module in every SOAP processing node acts as a safeguard to detect any rewriting attacks against SOAP Account along with attacks on other parts of the message.

To prevent this attack, the CheckSOAPAccount module will do some routine checks as soon as the SOAP message arrives. A first check is to make sure that the received SOAP message must have a SOAP Account header. If it is there then the module will verify the signature of the SOAP Account. If several intermediaries have their own SOAP Account then there will be a nested signature as it is described [1]. If verification is successful then the CheckSOAPAccount module will do the rest of the routine work as described in the section 5 of that paper [1]. Figure 10 and Figure 11 show a SOAP message having a SOAP Account as well as an attacked SOAP message showing an attempt to forge the SOAP Account header respectively. As in the previous example scenario, the attacker is introducing one new header and copying the SOAP Account information under the new header (Bogus) keeping the signature valid.

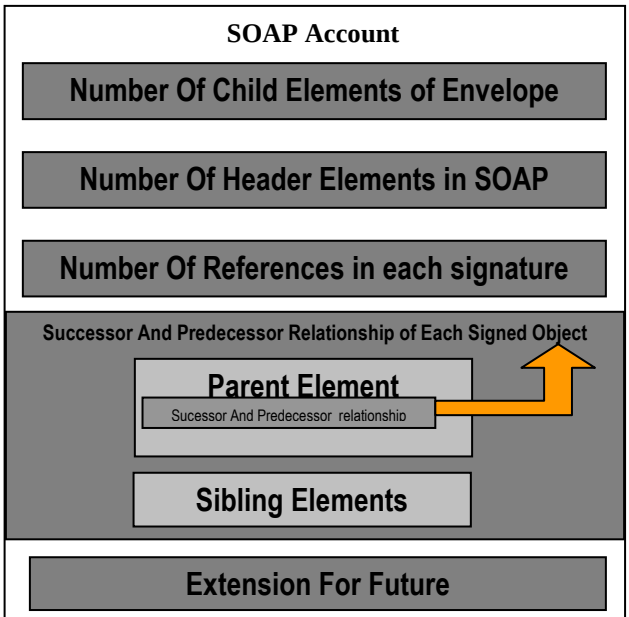

Figure 9: SOAP Account

But as we said that the CheckSOAPAccount module will check the presence of SOAP Account header as a SOAP header as soon as the message arrives. Since SOAP Account is copied under a new element it is not a SOAP header anymore (Figure 11). The module can immediately throw an exception saying that SOAP Account has been attacked. Again, we can detect the attack before doing any kind of computation intensive task like canonicalization.

Even if the attacker provides its own SOAP Account it will be immediately invalidated while doing SOAP Account signature validation. The reasoning behind this claim is as follows. Although the attacker may provide its own SOAP Account having updated SOAP structure information according to its attack, it can not provide its own signature key information to sign the SOAP Account in the existing <Security> header. The <Security> header contains legitimate key reference of the legitimate sender of the message (see Figure 10). In Figure 10, the legitimate sender is Alice who has provided her signature key reference in the <KeyInfo> element which will be used for signature validation. Besides, an attacked SOAP Account will be under a new false header (in the case of Figure 11 it is <BogusHeader>) which will be caught after the first routine check by the CheckSOAPAccount module. The attacker may insert a new <Security> header and its own key reference to validate the added SOAP Account. The CheckSOAPAccount module can det-

Figure 10: A SOAP message with SOAP Account before an attack (Excerpt)

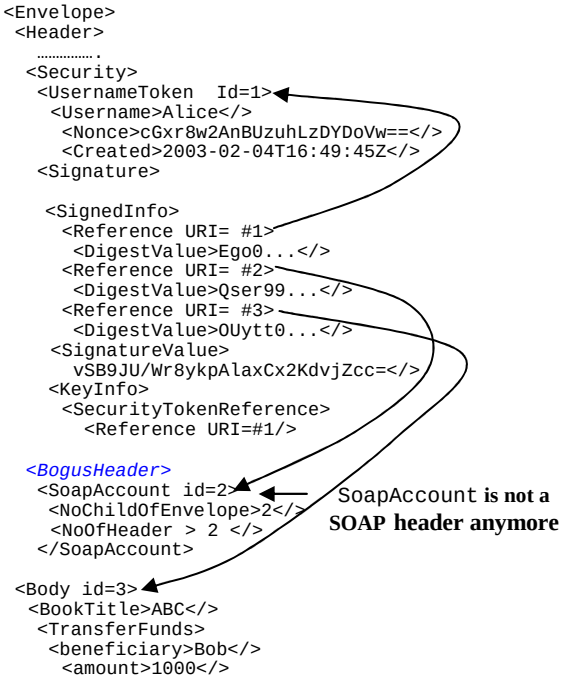

Figure 11. SOAP request after an attempt to attack on SOAP Account (Excerpt)

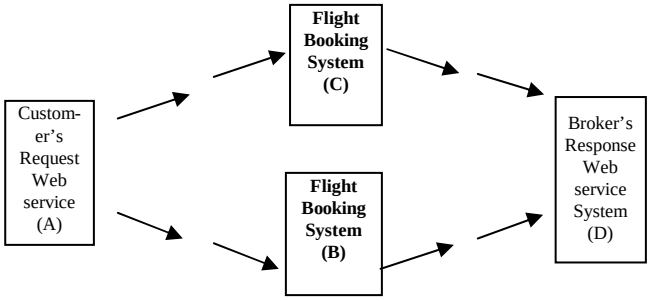

Figure 12: Travel Itinerary scenario

-ect this added key reference in the `<Security>` header in the same way described above. So, even if an updated `SOAP Account` is provided by the attacker, it will be detected eventually before signature value check of `SOAP Account`. Moreover, the nested signature feature of `SOAP Account` makes things harder for the attacker to forge the `SOAP Account`. How `SOAP Account` is processed using nested signature with several intermediaries is described in [1].

To understand the issue of forging `SOAP Account` with intermediaries and the reasoning to detect the attack, we consider the online travel itinerary scenario in the Figure 12 where there are several intermediaries, the sender, and the ultimate receiver of a SOAP message is shown. One service consumer (not shown in Figure 12) of a travel itinerary web service broker A, requests for a particular travel itinerary to get the best available price. The travel itinerary broker A may forward the same request several times to some flight booking systems of the related airlines (B, C). The broker is supposed to show the best available itinerary plan for the given request of the service consumer. A malicious SOAP processor (e.g. Broker, Flight Booking systems) may manipulate the SOAP message as in the previous scenario to present a bad itinerary plan in response. If the Broker is malicious it can temper with the itinerary request itself.

If B or C performs this malicious attack the consumer may not receive the best itinerary plan. In any case usage of `SOAP Account` information will allow us to detect the attack as soon the message is received and processed by the following SOAP processor's `CheckSOAPAccount` module. Here every SOAP processor will add its own `SOAP Account` using `AddSOAPAccount` module in a nested fashion [1] so that the ultimate receiver knows who did what. If any malicious attacker tries to forge the `SOAP Account` in the same fashion, the `CheckSOAPAccount` module of the following SOAP processor can detect the attack during

his routine checks of the validity of the received SOAP messages mentioned previously in this section.

6. Conclusion

In this paper we have presented a solution to protect SOAP messages against XML rewriting attacks. This solution was based on some prior work of ours [1] using SOAP message structure information, which we refer as `SOAP Account`, as an efficient technique to detect rewriting attacks. Since a `SOAP Account` might be a target of attackers itself, this paper focused on the preserving the integrity of a `SOAP Account`.

We have presented our analysis of protecting the `SOAP Account` from forging (XML rewriting attack) based on a real-world business scenario. We have concentrated on message level security and discussed two different message flows with and without using a `SOAP Account`. This was based on a simplified view of web service security in a SOA to show exactly where the concept of a `SOAP Account` fits into a SOA.

Considering that in a real-world scenario we might encounter systems with a payload of some hundred of thousands of SOAP messages exchanged on a daily basis, our earlier work on `SOAP Account` and XML processing related performance issues will need to be confirmed again, this time in the context of a more detailed performance analysis.

7. References

[1] Mohammad Ashiqur Rahaman., Rits Marten, Andreas Schaad, ”An Inline Approach for Secure SOAP Requests and EarlyValidation”,http://www.owasp.org/images/4/4b/AnInlineSOAPValidationApproach-MohammadAshiqurRahaman.pdf

[2] http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

[3] Bajaj, et al., *Web Services Policy Framework (WS-Policy)*, September,2004,http://www.ibm.com/developerworks/library/specification/ws-polfram

[4] K. Bhargavan, C. Fournet, A. Gordon, and G. O'Shea An Advisor for Web Services Security Policies, http:// research.microsoft.com/~adg/Publications/details.htm#sws05

[5] Microsoft Research; http://research.microsoft.com/projects/Samoa/

[6] K. Bhargavan, C. Fournet, and A. D. Gordon. Verifying policy-based security for web services. In *11th ACM Conference on Computer and Communications Security (CCS’04)*, pages 268–277, October 2004.

[7] T. Nadalin, ed., *Web Services Security Policy Language (WS-SecurityPolicy)*,Version 1.0, 18 December 2002, http://www.verisign.com/wss/WSSecurityPolicy.pdf

[8] K. Bhargavan, C. Fournet, A. D. Gordon, and R. Pucella. TulaFale: A security tool for web services. In *International Symposium on Formal Methods for Components and Objects (FMCO’03)*, LNCS. Springer, 2004

[9] G. Della-Libera, M. Gudgin, P. Hallam-Baker, M. Hondo, H. Granqvist, C. Kaler, H. Maruyama, M. McIntosh, A. Nadalin, N. Nagaratnam, R. Philpott, H. Prafullchandra, J. Shewchuk, D. Walter, and R. Zolfonoon. Web services security policy language (WS-SecurityPolicy), July 2005. Version 1.1.

[10] http://specs.xmlsoap.org/ws/2005/02/trust/WS-Trust.pdf

[11]http://specs.xmlsoap.org/ws/2005/02/sc/WSSecureConversation.pdf

[12] SOAP, http://www.w3.org/TR/soap/

[13] Carlos Gutierrez, Eduardo Fernandez-Medina, Mario Piattini “Web Services Enterprise Security Architecture: A Case Study” http://delivery.acm.org/10.1145/1110000/1103025/p10-gutierrez.pdf?key1=1103025&key2=9585273511&coll=ACM&dl=ACM&CFID=15151515&CFTOKEN=6184618

[14] G. Alonso and F. Casati and H. Kuno and V. Machiraju: Web Services: Concepts, Architectures and Applications, Springer-Verlag, 2004.

[15]http://java.sun.com/developer/technicalArticles/WebServices/soa2/SOATerms.html#soawhy

[16] http://www.ietf.org/rfc/rfc2246.txt

[17] http://www.ietf.org/rfc/rfc2818.txt