Protecting Data in Multi-Stakeholder Web Service Systems

Tan Phan, Jun Han, Garth Heward
Swinburne University of Technology
Melbourne, Australia
{tphan, jhan,gheward}@swin.edu.au

Steve Versteeg
CA Labs
Melbourne, Australia
Steve.Versteeg@ca.com

ABSTRACT

Current Web Service security standards have inadequate support for end-to-end protection of data when some receivers of the data are unknown to the sender. This paper presents an approach to aid collaborative partner services in properly protecting each other's data. Our approach allows each partner to derive an adequate protection mechanism with minimum performance overhead for each message it sends based on those of the corresponding messages it receives. We modify the message handling mechanisms of Web Service engines to dynamically gather protection requirements for a given outgoing message by aggregating requirements from original owners of message data.

Categories and Subject Descriptors
D.2.11 [**SOA**]: Service-oriented architecture

General Terms
Security, experimentation, theory, standardization

Keywords
Data security, protection mechanism, propagation

1. INTRODUCTION

Web Service-based SOA systems are often multi-stake-holder distributed systems that have multiple Web Service (WSs) which belong to different owners or stakeholders. One stakeholder often does not directly know every party that might have access to the data originally created by him. However, each party wants to protect the data generated/owned or relevant to him. WS-Security, which is the standard for Web Service security, allows for end-to-end protection of data between known parties. For example, if a party **A** sends sensitive data to a trusted business partner **B** via an un-secured network with potential untrusted intermediaries; WS-Security can be used to protect the data. However, if A does not know where its data flow to (e.g. **B** forwards the data to another partner **C** of **B** unknown by A) then WS-Security does not support **A** to specify protection requirements for its data between **B** and **C**.

While **A** might trust that **B** will be handle **A**'s data securely if **B** is aware of **A**'s requirements, currently there is no automatic mechanism for **B** to check **A**'s requirements on a given subset of **A**'s data. Besides, WS-Security is performance-wise expensive as encryption and canonicalization operations performed on XML-based SOAP messages add excessive overhead to message processing [1]. In the absence of **A**'s security requirements, **B** might have to blindly apply full protection for **A**'s data which has a heavy performance penalty, or blindly use non-secure channels, in which case **A**'s sensitive data might be under-protected. From both **A** and **B**'s point of view, it is desirable that there exists a mechanism that enables **B** to derive **A**'s requirements for **A**'s data that **B** is sending out. The same can be said for **C** which receives **A**'s data through **B** or any other parties where **A**'s data flow to. The assumption here is that **A** trusts **B** will not deliberately use the data in a malicious way (because **A** knows **B** or **A** and **B** have partnership). However, **A** wants to prevent **B** from accidentally doing so (i.e. sending out **A**'s sensitive data without using a secure channel) by not being aware of **A**'s protection requirements of its data. This paper proposes a mechanism to make **B** aware of **A**'s security requirements and propagate such requirements to **B**'s partners (**C**/others) with the flow of **A**'s data.

2. PROPAGATING DATA PROTECTION REQUIREMENTS

Our approach to address propagation of data protection requirements is to enable each service to derive adequate protection mechanisms for output data based on those of input data. Each party maintains the dependency between input and output of its services so that the protection requirements for the output can be computed from those of original inputs. We use a mechanism of *content-based message correlation* (i.e. using application semantics) to dynamically correlate different messages related to a single business scenario. We assume that the correlation information is specified at design time (static correlation) and support runtime establishment of security requirements of output messages based on the requirements of input messages using this correlation information.

A protection mechanism *PM* for data is determined based on a combination of the encryption and digital signature protection for the data, $PM = enc \wedge sig$ where *enc* or *sig* can potentially be empty. We define a ranking scheme for protection mechanisms based on their strength and a set of techniques for combining protection mechanisms together. For example, an encryption mechanism is defined as a tuple $enc(b, ea, kl)$ where *b* is the encryption binding method, (symmetric or asymmetric), *ea* is the encryption algorithm and *kl* is the length (number of bits) of the key used in the encryption. Based on that, we calculate that an encryption mechanism $enc_1(b_1, ea_1, kl_1)$ is as or more secure than an encryption mechanism $enc_2(b_2, ea_2, kl_2)$, $enc_1 \geq enc_2$, if $b_1 = b_2 \wedge ea_1 = ea_2 \wedge kl_1 \geq kl_2$.

At each service, the required protection level for a message is determined as the union of original senders' requirements and the service's own requirements. For each message being sent, the predefined message correlation information is looked up to identify the original corresponding input messages, their senders and the original senders' protection requirements. The requirements are then ranked and the strictest requirement is selected in case there is an overlap. If the requirements mandate different types of mechanisms, the requirements are combined together. The service then calculates the final required protection level for the outgoing message by combining the calculated requirements with its own requirements on the data carried in the message. The service then, based on the calculated required protection level, automatically selects an appropriate security channel on which the message will be delivered. This channel is the one that satisfies all the protection requirements with the least level of over-protection. If an outgoing message needs protection but no secure channel is available, it will be rejected from being sent. This mechanism can be used by a Web Service infrastructure (e.g. Axis2) and Web Service security infrastructure (e.g. Apache Rampart) for providing data security as demonstrated by our prototype, described below.

Each node in a service-based system dynamically propagates security requirements to its partners as part of the message flow. This ensures that a given party's data is always either delivered using a channel at least as secure as the original channel, or not delivered at all. The advantage of this approach is that it allows for the dynamic propagation of security requirements with the message wherever the message is sent, without the need for a centralized controller or a known system topology. This is essential for SOA because of the nature of dynamic composition/binding of element services in SOA systems.

3. PROTOTYPE IMPLEMENTATION AND PERFORMANCE EVALUATION

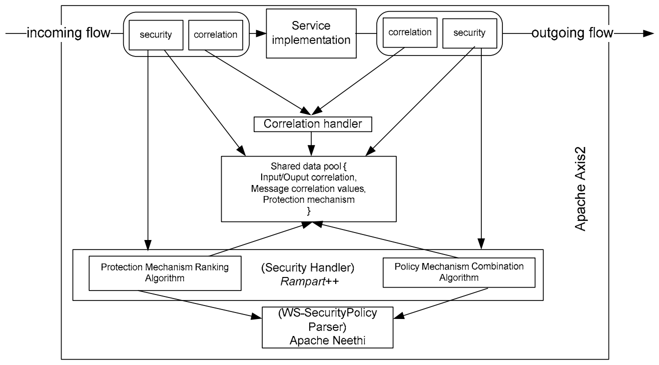
incoming flow
security
correlation
Service implementation
correlation
security
outgoing flow
Correlation handler
Shared data pool { Input/Ouput correlation, Message correlation values, Protection mechanism }
Apache Axis2
Protection Mechanism Ranking Algorithm
(Security Handler) Rampart++
Policy Mechanism Combination Algorithm
(WS-SecurityPolicy Parser) Apache Neethi

Figure 1. Prototype architecture and deployment

Prototype: In contrast to ensuring third-party message security by using maximum security all the time, our approach of dynamic security seeks to achieve higher efficiency. We implemented a prototype for our approach based on Axis2 Web Service engine and the Apache Rampart WS-Security infrastructure. Figure 1 depicts the architecture and deployment of the prototype. The protection mechanism ranking and combination techniques are deployed as an add-on to Rampart (we call it *Rampart++*). An Axis2 module called the *correlation handler* is implemented to handle correlation and message identification. This module uses the inflow and outflow message processing mechanism of Axis2 to extract relevant details related to message protection mechanisms and correlation information from incoming messages. It stores them in a shared data pool accessible by inflow and outflow handlers so that outflow handlers can retrieve the necessary protection requirements and message identification. The message processing is transparent to the implementation of the service and can be enabled/disabled without affecting the service's functional behavior.

Performance evaluation. We have implemented an example system that has three services **A, B**, and **C** and deployed them on a local network with a Gigabit Internet connection. **A** sends **B** messages with sensitive data in the form of *scans* of personal documents, which **B** forwards to **C**. We varied the size of the *scans* from around 1kb to around 50 kb in our experiments to show the effect of different message sizes. There are three security channels for the connection between **B** and **C**, one with *full protection* (both encryption and signing are applied), one with only signing (*sign only)*, and one with *no protection*. We changed the requirements of **A** and **B** during the experiment so that *Rampart++* will select one of the three channels respectively. For the purpose of showing the overhead incurred by our approach, we have also evaluated a scenario when our system is disabled and full protection is blindly applied. Each test case executed 100 times. The result is shown in Figure 2.

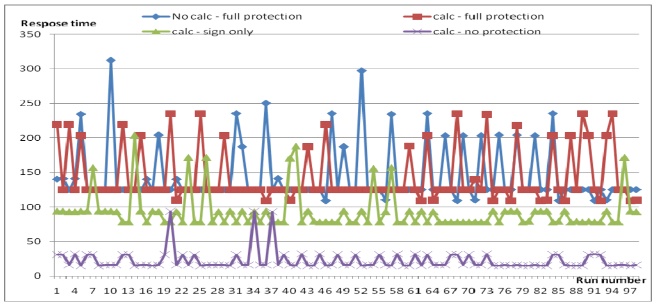

Figure 2. B-C connection's response time in different settings

In Figure 2 "calc" means *Rampart++* is engaged while "no calc" means the module is disabled. It can be seen from Figure 2 that the overhead incurred by our approach is relatively small (the lines of "No calc – full protection" and "calc – full protection" are almost identical). It also shows that when our approach detects that full protection is not needed, but only a digital signature is needed, it can improve the response time by an average of around 33% compared with blind full protection. In the case when no security is needed, the improvement is significant; the response time is on average 600% faster compared with the case when full protection is blindly applied.

4. CONCLUSIONS

We have presented an approach that enables collaborative service partners to protect data in transit according to the requirements of parties who created and processed that data. Our approach ensures that data is not under-protected while not having to blindly employ the most secure mechanism possible. The main contributions of our work include 1) a method for calculating the protection requirements for a service's output messages from those of input messages and the service's own requirements and 2) a mechanism for propagating the requirements together with the flow of the data in the system.

5. REFERENCES

[1] M. Juric, I. Rozman, B. Brumen, M. Colnaric, and M. Hericko, "Comparison of performance of Web services, WS-Security, RMI, and RMI–SSL," *The Journal of Systems & Software,* vol. 79, pp. 689-700, 2006.