Abstract

Security certification assesses the security posture of a software system to verify its compliance with diverse, pre-specified security controls identified by guidelines from NIST and the US Department of Defense. Service-oriented architectures (SOA) are difficult to certify because they require compliance verification over a mix of local, global, and interaction criteria dictated by the policies of the participating services and SOA governance. Web services further contribute to this difficulty because they lack direct methods to express security controls. Besides being understandable, the method of expression should indicate potential problems complying with chosen services. This paper presents a method for configuring of web service standards to enforce security requirements on service interaction specification documents within a SOA. The outcome serves as a mechanism to direct the population of constraints derived from security controls within standards specification documents, such as WS-Policy. We focus on security controls for auditing and how these can be enforced in an SOA. We introduce a reusable architecture to notate the comparison of security controls across services.

1. Introduction

Web services and Service-Oriented Architectures (SOAs) have promoted rapid development of integrated systems that span traditional boundaries. Web services are built on standardized interface specifications to facilitate their integration and efficient execution. Businesses use these specifications to orchestrate workflows, combining internal and external services to form their SOAs.

Security remains an issue in SOAs. To certify that web services and their interactions comply with stated security controls, the controls must be represented in a manner in which their enforcement is easily established or verified in a SOA. For federal information systems, NIST defines a set of security controls in its SP800-53 that are requirements for certification. For US Department of Defense information systems, the DoD 8500.2 outlines Information Assurance (IA) controls as security requirements for certification. As SOAs proliferate, they must meet these certification demands. Though web services use XML document specifications that can express certain security constraints, in many cases a security policy needs to employ multiple standards. This need raises the issue of maintaining consistent representations within and among the document specifications.

Multiple problems emerge when determining if a SOA abides by a set of security and IA controls. It is difficult to capture the security posture of loosely coupled services in the SOA and evaluate the associated risks [1]. When mandated, security controls, such as audit, pose unique enforcement problems because web services must incorporate specific logging and transition capabilities into their functionality [2]. Once web services are orchestrated into a flow, certifying the SOA can manifest risks that only major architecture changes will resolve. Since there often exist multiple, redundant web services by different vendors, it is prudent to perform a design time analysis to uncover critical security compliance violations. Then, designers can choose web services that allow for a more secure SOA.

In this paper, we employ web service standards to promote known security intents of the service and its related interaction behavior. We abstract security controls from the NIST and DoD documents, mapping them to explicit elements in and across standards specifications. We advocate a reusable architecture that notates the connectivity among standards when enforcing a particular control or group of controls. The objective is to empower the SOA developer to choose services that meet designated security certification criteria and easily identify potential violations of those criteria. We use a group of audit controls as an example.

2. Background
The NIST SP800-53 [3] and DoD 8500.2 [4] state certification guidelines for security controls that organizations choose for their system. The security certification process ensures the resulting system complies with those controls. Thus, organizations can be flexible when securing different system types and consistent when securing systems of the same product line [5]. Previous research maps the text of the controls to risk components (assets, criticality, etc.) to correlate with control categories [6].

Researchers examine SOA security issues by focusing on mitigation strategies such as determining the existence of a violation by employing models to understand how assets traverse services [1]. Security patterns match detected problems with mitigation design strategies. Complex security problems may require applying multiple security patterns. Dong et al., [8] use a model checker to examine composed security patterns for accuracy. Their model checker determines if the patterns retain core mitigation properties once composed. To apply the patterns, they interpret the architecture requirements surrounding the potential security problem. They translate these requirements to UML and compare them with composed patterns for consistency. The difference is that there is no formal method to get from the security control to security problem.

The Service Matchmaker [9] uses an ontology-based approach to extend the WS-Policy standards document with semantics. The goal is to determine if two policy documents match. The claim is that without semantics, this matching produces limited results. The authors do not extend the semantics or match to the inter-referencing problem of the standards documents that have complementary information.

3. WS-* Specifications

We exemplify WS standards specifications, commonly referred to as WS-*, for security in column 1 of Table 1. Because WS-* play a significant role in expressing the web service security posture, they are essential to the assessment of SOA compliance. An advantage of the standards is that they have consistent syntax, structure, and semantics. A downside is that they are complex and cross-referential that can lead to inconsistencies in their instantiations. Goal statements for each standard are shown in column 2 of Table 1 as taken from the respective websites of OASIS or W3C [10-11]. Because XML is extensible, several of standards consist of multipart specifications with extensions; each of which adds unique features. The last column in Table 1 includes relevant XML elements that each specification introduces. Their inter-referencing, shown next, makes it difficult to guarantee that they are in agreement. Manifesting the security properties of a service requires deep understanding of the standards specifications.

Figure 1 illustrates the hierarchical relationship between each standard and its relation to the core web services architecture that contributes to the inter-referring among the specifications. Web services are described via the Web Services Definition Language (WSDL in Table 1) [12]. Service bindings from wsdl:service attach specific endpoint address information to the exposed interfaces. This information is necessary for the service to become real and executable.

Web services communicate by exchanging SOAP messages, the contents of which are governed by a service’s WSDL document (row 3 of Table 1). The WSDL specifies messaging expectations of a web service by defining schemas for SOAP messages in wsdl:types. This designates the specific assets that the service expects to interact with. Therefore SOAP messages must correspond to the WSDL interface guidelines the web service publishes. Because WSDL documentation describes the publically exposed interface of web services, internal assets are not found in the interface specification. The owner of the service must know what assets to expose to the security architecture for compliance to be checked. Message types are attached to service interfaces (wsdl:interface) and the exposed operations (wsdl:operation).

Each operation defines a set of messages that are sequentially transferred at the service interface. WSDL defines a set of corresponding Message Exchange Patterns (MEP), such as In, Out, In-Out, and Out-In that services can use to describe common communication styles. Each MEP defines the order in which specific SOAP messages leave or enter the web service interface. This specification of message exchange is bound to a single service WSDL, for example stating that a SOAP message of particular type leaves a service but not stipulating where it goes to.

In SOAP, messages are wrapped in an envelope (env:Envelope) that contains a message with a payload (env:Body) and a corresponding header (env:Header). The message header describes the internal structure of the message and embeds information about how the SOAP message is encoded for transfer between services. The core specification of SOAP does not specify how message security is implemented, only the envelope for delivering messages once crafted.
Table 1. WS-* Standards Goals and Key Elements

<table>
<thead>
<tr>
<th>Specification</th>
<th>Specification Goal</th>
<th>Key XML Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSDL</td>
<td>“This specification defines the core language which can be used to describe Web services based on an abstract model of what the service offers.”</td>
<td>wsd:types&lt;br&gt;wsd:interface&lt;br&gt;wsd:operation</td>
</tr>
<tr>
<td>WSCL</td>
<td>“WSCL allows the abstract interfaces of Web services, i.e. the business level conversations or public processes supported by a Web service, to be defined…”</td>
<td>wsc:Conversation&lt;br&gt;wsc:ConversationInteractions&lt;br&gt;wsc:ConversationTransitions</td>
</tr>
<tr>
<td>SOAP</td>
<td>“SOAP Version 1.2 provides the definition of the XML-based information which can be used for exchanging structured and typed information between peers in a decentralized, distributed environment.”</td>
<td>env:Envelope&lt;br&gt;env:Header&lt;br&gt;env:Body</td>
</tr>
<tr>
<td>WS-Security</td>
<td>“The goal of this specification is to enable applications to conduct secure SOAP message exchanges. This specification is intended to provide a flexible set of mechanisms that can be used to construct a range of security protocols…”</td>
<td>wss:Security</td>
</tr>
<tr>
<td>WS-Policy</td>
<td>“The goal of WS-Policy is to provide the mechanisms needed to enable Web services applications to specify policy information.”</td>
<td>wsp:Policy</td>
</tr>
<tr>
<td>WS-Secure Conversation</td>
<td>“This specification defines extensions to allow security context establishment and sharing, and session key derivation. This allows contexts to be established and potentially more efficient keys or new key material to be exchanged, thereby increasing the overall performance and security of the subsequent exchanges.”</td>
<td>wsc:SecurityContextToken</td>
</tr>
<tr>
<td>WS-Trust</td>
<td>“In this specification we define extensions to WS-Security that provide: Methods for issuing, renewing, and validating security tokens.”</td>
<td>wst:RequestSecurityToken&lt;br&gt;wst:RequestSecurityTokenResponse</td>
</tr>
<tr>
<td>WS-SecurityPolicy</td>
<td>“This document defines a set of security policy assertions for use with the WS-Policy framework with respect to security features provided in WS-S SOAP Message Security, WS-Trust, and WS-SecureConversation.”</td>
<td>sp:EncryptBeforeSigning&lt;br&gt;sp:SignedParts&lt;br&gt;sp:EncryptedParts</td>
</tr>
</tbody>
</table>

Encoding the sequential passing of messages between services requires additional WS-* languages. The Web Services Conversation Language (WSCL in Table 1) links multiple web service together by encoding interactions (wscl:ConversationInteractions) referencing SOAP messages and the order they are passed between services. Other languages, such as the Business Process Execution Language (BPEL) for example, accomplish similar functionality but are tied to a specific execution domain. WSCL specifies the exchange of SOAP messages between services in using a minimal amount of XML to reference schemas for messages.

WS-Security provides structure for SOAP message security by notating message-level security protocols through extension of the core SOAP specification. WS-Security encodes attributes about security features (e.g. message signature, payload encryption, authentication tokens) through the specification of a security construct (wss:Security) that is embedded within env:Envelope. WS-Security uses XML-Encryption, XML-Signature, and various XML token expression specifications (username, X.509, etc.) to provide a basic set of security functionality within SOAP message headers. Other extensions to WS-Security such as WS-Trust and WS-SecureConversation provide advanced security features (last column, Table 1) such as the ability to broker tokens across system boundaries (wst:RequestSecurityToken) and the establishment of secure session contexts (wsc:SecurityContextToken). Some of these extensions require extra SOAP message exchanges prior to regular message exchange in order to derive the necessary SOAP X.509 key information.

**Figure 1. WS-* standards relationships**
WS-Policy is a multipart documentation language similar to WSDL. WS-Policy describes the functionality of a web service to support specific WS-Security mechanisms. It describes policy alternatives that can be used by invoking clients offering security over messages, specific operations, or entire interfaces. Although WS-Security describes the security modifications to the SOAP messages that a web services exchanges, the service must be encoded to support those protocols. The WS-Policy documentation notates in wsp:Policy the security features that a web service supports. WS-SecurityPolicy, as part of WS-Policy, includes the elements for specifying security capabilities such as sp:SignedParts that explicitly defines parts of a SOAP messages that must be digitally signed prior to sending. Other elements such as sp:EncryptBeforeSigning designate protocols for how to enact the policy where ambiguity may exist.

Policy attributes may be defined for web services at three different levels of attachment: message, operation, or endpoint. For this paper our analysis focuses on endpoint policy specifications as they can be generated for each interface the WS exposes.

4. SOA System Instantiation

For compliance assessment against known certification criteria, knowledge of the services and their configuration must be captured. For a SOA, we rely on the WS-* standards in Section 3 to jointly express that embed the system configuration. Thus, the certifier must be directed to the particular XML elements that hold key information of interest. Using the elements from Table 1, and the discussion of their interrelationships in Section 3, we show how they can be formulated to express security controls for direct certification.

We target audit security controls related to web services due to their complexity and relationship to other policy groups, such as the requirement for message protection. Auditing for web services can refer to either the logging of events internal to the execution of the service, such as database transactions, or the logging of the service and its execution within a corresponding application server. Logging these events can generate several different types of audit records including those that track authentication, access, authorization, runtime performance, and web service faults (exceptions) [2].

Table 2 overviews denoted security controls. We translate system level constraints within the controls to XML requirements in specific WS-* standards. The specific element constraints are named and reveal the design of the web services architecture to assess for certification compliance. For example, the first row in Table 2 summarizes the NIST criteria (SP800-53 AU-2(1)) for audit controls stipulating that services must support a common data type for audit records.

<table>
<thead>
<tr>
<th>Table 2. WS-* Configuration Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security Control</td>
</tr>
</tbody>
</table>
| Policy – NIST 800-53 AU-2(1) | System compiles audit records from multiple components into a systemwide (logical or physical), time-correlated audit trail. | • Services support interactions referencing a common audit data type | WSDL (all services) | Across all services wsd:types includes a dedicated schema for an audit record. Example: 
  <wsp:Policy wsu:Id="policyID"/>
  <sp:Body/>
  <sp:PolicyReference URI="#auditRecord"/>
| | | • A service exists as Audit Collector | WSDL (single service) | At most one web service with the following: 
  <service name="AuditCollector" interface="auditingInterface"/>
| Derived Architecture Configuration | System components direct audit records to a Audit Collector | • Audit Collector supports in-bound communication | WSDL for Audit Collector | <wsp:interfaceName="loggingInterface"/>
  <wsp:operationName="logRecord"/>
  <wsp:input.../>
| | | • System services support out-bound communication | WSDL for all other services | <wsp:interfaceName="LoggingInterface"/>
  <wsp:operationName="generateLog"/>
  <wsp:output.../>
| Policy – NIST 800-53 AU-9(3) | System uses cryptographic mechanisms to protect the integrity of audit information and audit tools. | • WSDL Specifications for each service have an associated WS-Policy | WSDL (all service) | <wsp:PolicyReference URI="#PolicyID"/>
| | | • Audit records are digitally signed to protect integrity | WS-Policy | <wsp:Policy wsu:Id="PolicyID"/>
  <sp:SignedParts>
  <sp:Body />
| Policy – DoD 8500.2 ETCP-1 | The contents of audit trails are protected against unauthorized access, modification or deletion. | • Encryption is applied to audit records in transit | WS-Policy | <wsp:Policy wsu:Id="PolicyID"/>
  <sp:EncryptedParts>
  <sp:Body />
This translates to requiring the existence of a corresponding WSDL data type for all services.

As increasing knowledge about the SOA specification occurs, more specific XML elements can be named. This knowledge introduces complexity in the overall representation of the security controls due to the referential nature of each of the WS-* specifications. As indicated in Table 2, to perform the function of a central audit collector, a single service must have a WSDL specification (AU-2(1)) indicating an auditingInterface. As more security controls are added, and more of the overall SOA configuration can be derived, elements within and across documents must retain a high degree of compatibility. The need for an audit collector that supports in-bound communication (row 3 in Table 2) similarly requires the auditingInterface from the prior control statement to both match in XML as well as expose an inbound WSDL operation (wsdl:input).

Although these requirements are internal to a single document type, constraints can exist between document types, as shown by the last two rows of Table 2 for the controls AU-9(3) and DoD 8500.2 ETCP-1. Each stipulates that the system must use cryptographic controls to protect the integrity and confidentiality of audit records in transit. To comply with these controls, a corresponding WS-Policy constraint must attach SOAP level signatures and encryption to the audit records in transit. WS-Policy encodes these in XML as the corresponding sp:SignedParts and sp:EncryptedParts of the wsp:Policy element. However, a match between the WS-Policy wsu:Id and the referencing wsp:PolicyReference URI of the WSDL-documents must exist (policyID, as indicated by underlined text) otherwise the attached policies do not govern over the services.

Incorporating other WS-* specifications to derive more XML constraints requires a balance between selecting the appropriate security controls and how the derived architecture is configured. Many WS-* standards exist as extensions to the core set of security functionality offered by WS-Security. For example, within Table 2, WS-Policy includes references to WS-SecurityPolicy (as indicated by the sp namespace reference in sp:SignedParts and sp:EncryptedParts). To further the example, if the SOA under certification requires federated interactions (combinations of services across different systems), a constraint interpretation can include the requirement that authentication tokens are exchanged according to the specification of WS-Trust. These types of XML constraints are represented via expanding associated WS-Policy documents. However, other specifications such as WS-SecureConversation offer completely new constructs for service interaction. This case requires adding new documents from those listed in Table 1 to advance the XML constraint requirements.

5. Modeling Perspective

We augment a basic software architecture diagram with configuration artifacts that notate the position and flow of security entities with respect to where/how they must be assessed for compliance. The abstraction of the SOA that displays where WS-* standards are instantiated with respect to the service interface. Internal specification details are needed for full verification. Then using the internal cross-referencing, certification problems are detectable with corresponding XML elements. The diagram expands to accommodate the increasing demand for system security properties. It consists of:

- **system components or services**: the functions or operations the service offers
- **ports or communication interfaces**: what the service exposes that the WS-* documents are expected to enforce
- **system assets**: what should be protected from a security standpoint
- **interaction requirements**: governed by criteria statements indicating the need for asset transfer between services
- **communication requirements**: connector or channel constraints on communication from security control requirements [7]

Figure 2 shows the basic architecture. Components or services expose their interface at certain ports to facilitate the transfer of assets through the system. The arrow designates the directional flow of an asset as it is transferred across component boundaries due to a governing policy or system requirement. The port may have policy restrictions that dictate the conditions (e.g., a data protection encryption mechanism) for the exchange. The security mechanism is associated with the receiving component where it applies its own policy expectations to the transfer and the asset.
Figure 2. Security Architecture

We show in Figure 3 a WSDL specification for two web services along with an associated WS-Policy document for message encryption and signature generation. We focus on only on the XML entities already discussed in 4 and Table 2. In the example ServiceA is a system service that must adhere to the SOA security control guidelines for secure audit trail generation and transfer to a centralized AuditCollector. An outbound message comes from a LoggingInterface that generates a message adhering to the respective XML Schema declaration. The service specification for the AuditCollector includes an auditingInterface that accepts the XML audit record.

The architecture diagram in Figure 4 makes specific reference to XML elements where audit information passes through the service’s as specified in the respective WSDL interfaces. Attached to each interface is the WS-Policy specifications for SOAP message encryption and the corresponding security control from which the requirement is derived.

The diagram notation can be extended to accommodate additional XML constraints that map directly back to a WS-* document. However further research is necessary to arrive at a complete breadth of understanding about the specification of a SOA and its security as expressed through a complete set of relevant XML elements and their cross document connectivity.

7. Conclusion

In this paper, we illustrate the complexity of expressing NIST and DoD security controls in web services standards specifications in order to certify SOAs. We show the potential for proper representation and referencing by explicitly relating security control properties to specific WS-* XML elements. An architecture diagram embodies these essential elements as well as the designated asset and flow to provide a global perspective of the system relative to its compliance with security controls. With this approach, our future work will examine other policies, such as authorization and contingency planning, and their impact on WS-* specifications.
Figure 4. Auditing WS-* Security Architecture

Acknowledgement. This material is based on research sponsored in part by the Air Force Office of Scientific Research (AFOSR), under agreement number FA-9550-09-1-0409 and in part by the Air Force Research Laboratory (AFRL) under agreement number FA8750-10-2-0143. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied of the AFOSR, AFRL, or the U.S. Government.

8. References