Security as a Service - A Reference Architecture for SOA Security

Mukhtiar Memon, Michael Hafner and Ruth Breu

University of Innsbruck, Austria

Abstract. Securing service-oriented systems is challenging, because like business services the security services are equally distributed in SOA systems. Enforcing security exclusively at the endpoints creates a significant security burden. Also, every endpoint has to implement the entire security infrastructure, which is an expensive approach. Currently, there is very little work done to separate security from service endpoints. We propose a *Security As A Service* (SAAS) approach, which shifts major security burden from service endpoints to dedicated and shared security services within a security domain. Security services are composed from components, and integrated based on the Service Component Architecture (SCA) model. In this contribution, we apply the SAAS paradigm to implement security for SECTISSIMO, which is a platform-independent framework for security modeling and implementation [9]. [1]

1 Introduction

Security is a complex non-functional requirement of SOA systems, which are composed from services deployed across different locations and diverse platforms. The main problem for SOA security is the lack of security modeling frameworks, based on consistent and formal methods. We have addressed this problem in [9], and proposed a solution as SECTISSIMO security modeling framework.

The second most important aspect, which is the focus of this contribution, is the architecture for SOA security. From this point of view, the available architectures and supporting standards are still in the nascent state. SOA system deals with many facets of security, ranging from basic encryption and access control to security monitoring and management across domains. A domain basically comprises a number of service endpoints (or simply endpoints), which offer atomic or composed services. The workflow formed by composed services spans multiple endpoints in different domains. Therefore, the security provided at a particular endpoint may not be sufficient to secure all the services in the composition. The reason is obvious, as there is no common point, which all the endpoints can rely for security assessment and evaluation. An endpoint can perform only primitive security tasks such as encryption, signature etc. While, for complex security tasks, an endpoint has to rely upon the decisions based on the security

[1] This work was partially supported by the SecureChange (ICT-FET-231101) EU project and the SECTISSIMO (P-20388) FWF project.

Memon M., Hafner M. and Breu R. (2009).
Security as a Service - A Reference Architecture for SOA Security.
In *Proceedings of the 7th International Workshop on Security in Information Systems*, pages 79-89
DOI: 10.5220/0002174900790089

protocols traversing other domains. This necessitates a hybrid approach combining integrated and decoupled security. The integrated *Endpoint Security* takes care of primitive security tasks and the decoupled security, which is shared by all the endpoints within a particular domain, handles more complex security tasks.
On the grounds of these facts we present an approach for engineering security-critical SOAs, consisting of two phases:

1. *Phase 1*: Modeling of security together with functional modeling and generation of security artifacts through semi-automated transformation.
2. *Phase 2*: Implementation of security as a service, using *Service Component Architecture* (SCA) model.

We have worked out the model-driven approach for security in the SECTET [5] and SECTISSIMO [9] frameworks. The contribution of this paper is our proposal to deploy security services within a domain, which are shared by all the service endpoints in that domain. These services are capable of executing security protocols among different security domains, thereby reducing the major security burden of service endpoints.

The remaining paper is organized as follows. *Section* 2 presents related approaches. Based on an example use case, the security requirements of healthcare system are described in Section 3. *Section* 4 sketches the SECTISSIMO security framework. *Section* 5 presents limitations of endpoint security in the perspective of SOA. Proposed Security As A Service (SAAS) approach is discussed in Section 6, whereas, *Section* 7 presents our solution as a Reference Architecture for SOA Security. The implementation of the solution based on SCA is discussed in *Section* 8, followed by challenges related to SAAS approach in *Section* 9. Concluding remarks and future work are given in *Section* 10.

2 Related Work

This paper mainly focuses on a reference architecture for security-critical SOA systems based on *Security As A Service* (SAAS) paradigm. In [4], Heather Hinton et. al. have introduced the SAAS approach and proposed a Security Decision Service (SDS), which provides service-based PDP to multiple enforcement points. A more comprehensive discussion is given in [6] by R. Kanneganti and P. Chodavarapu, for implementing authentication, trust and secure conversation as separate services to solve security manageability and interoperability problems. Identity Externalization issues for service-oriented security (SOS) are presented by Oracle in [15]. In [7], J. Lopez uses dedicated security services for advanced authentication and authorization requirements. G. Petterson [17] presents security-specific views for SOS architectures. Antivirus products use the SAAS approach based on security services for small and medium businesses [8].

These approaches basically discuss separation of security from endpoints for individual security services but this does not solve the key security problems of SOA, which includes complex security requirements such as non-repudiation, security compliance and monitoring. Also, these approaches do not clearly define how security tasks should be split between endpoints and the shared security. We have investigated these problems

very carefully, taking into account a more comprehensive set of security services.

There are other even more important aspects which have so far not been addressed. For example, none of the above approaches define how security services depend on each other and how to configure security at deployment-time. Without defining the dependency and relationship among security services, it is not possible to design complex security protocols among various endpoints of different domains. An other aspect that adds value to our approach is the reusability of security components. With our approach it is possible to implement same security components with different deployment-time configurations. To achieve these values, we compose the components based on Service Component Architecture (SCA) [16]. We use SCA's *Dependency Injection* mechanism to define dependency among the components. Similarly, we use SCA's component *Properties* for deployment-time configurations of security components (Details in Section 8).

3 Motivating Example

We illustrate our concepts with an industrial use case of a healthcare system, which consists of medical services provided by various stakeholders such as hospital, radiography, pharmacy and insurance [2]. The security workflows for cross-domain authentication, authorization, non-repudiation and monitoring etc. execute the protocols among multiple domains of these stakeholders.

Fig. 1. Distributed Healthcare Scenario.

We consider two domains, represented by Hospitals 1 and 2 respectively, as shown in Figure 1. The users in these domains can be authenticated with the local identities assigned to them in the form of credentials such as username password and digital certificate. In this scenario, a practitioner from Hospital 1, accesses a service i.e. *viewRadiography* from Hospital 2. We assume that she possesses the local identity for Hospital 1, but this identity is not valid for Hospital 2, because every hospital maintains the local identities of its own domain users. What is required here, is another (security) service, which federates the practitioner's identity between two hospitals. One of the critical questions could be that which of the service endpoints in the hospitals should perform the task of identity federation and why. This is just one example of a security task between two domains, but in the real world there are a number of security tasks such as authorization decisions, privacy enforcement, security protocol execution, logging, security compliance verification and security monitoring among many healthcare stakeholders. If these tasks are assigned to a specific service endpoint, it will not only

create additional processing burden, but also create interoperability problems. To address these problems, we propose to free the service endpoints from security tasks (as much as possible) and assign them to the dedicated and shared security services.

4 SECTISSIMO Framework

The proposed SAAS approach is used to design the reference architecture for SECTISSIMO framework. SECTISSIMO provides a layered approach to model security requirements in parallel with functional modeling and generate security artifacts using transformation. The security requirements are modeled in a platform-independent way, using abstract security protocols and controls. In the second phase, platform-specific security artifacts (i.e. security policies) are generated from the abstract models based on supporting security infrastructures of the target platform. The security services in the SAAS component (Section 6.1) of the proposed reference architecture execute these platform-specific artifacts which are generated from models. For detailed discussion about SECTISSIMO, please refer [9].

5 Limitations of Endpoint Security

The *Endpoint Security* is based on the assumption that all the security infrastructure components are integrated with the service endpoint. This includes identities stores, policy repositories, protocol execution engines and monitoring agents. Integrating them with the endpoint is an expensive solution, because the service endpoint administration has to deploy and configure all the required hardware, network and applications. It also increases the responsibilities and security burden of the endpoints, which could slow down its performance. Additionally, this creates the problems for security interoperability as different domains implement different standards, mechanisms and configurations for security. The approach is quite effective for primitive security tasks such as encryption, signature and time-stamping etc. But, it is not applicable for SOA security, in which services and security controls may not be placed exclusively at the same endpoint. These limitations markedly advance our understanding that endpoint security leads to some architectural and interoperability problems pertaining to SOA security.

6 Security As A Service (SAAS) Approach

In this section we define our approach and present our arguments why the SAAS approach is a better choice to solve SOA security as compared with endpoint security.

The Security As A Service (SAAS) approach can be defined as an architectural solution for SOA security, based on decoupled and shared security services within a domain.

If we examine the security requirements of a service endpoint, we will see that a service endpoint is mostly concerned with the security decision (e.g. token validation, authorization), that results from a security protocol. For instance, the decision that asserts if the security tokens of a Practitioner are valid or if she has certain permissions to

call a service. As a further step, the endpoint enforces that security decision. We assume that the task of executing the security protocol and communicating the decision to the target endpoint should be performed by separate security services.

Figure 2 shows the architecture of shared security services based on the proposed

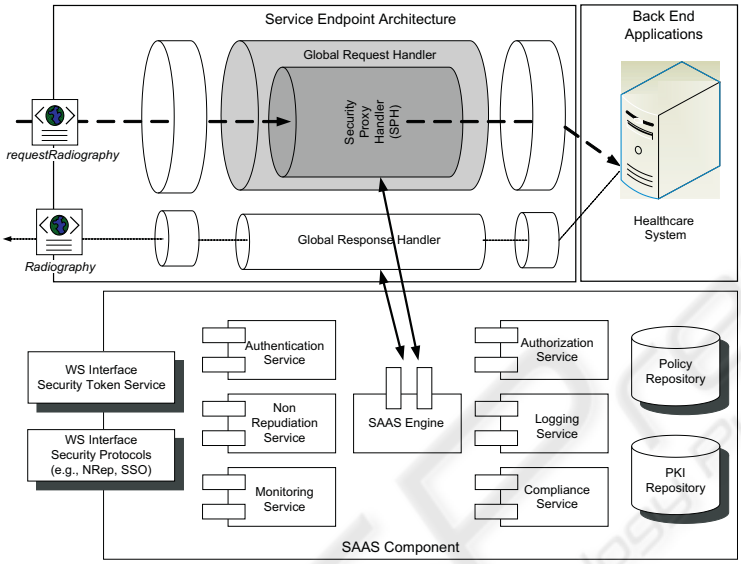

Fig. 2. The Architecture of Shared Security Services in a domain.

SAAS approach. The upper part shows the *Healthcare System*, which comprises of various service endpoints. The Global *Request* and *Response* Handlers are integrated with the service endpoint. The handlers intercept the incoming and outgoing messages to/from a service endpoint and provide primitive security. For example, in Figure 2, the Request Handler intercepts a Practitioner's request to access the *viewRadiography* service and the Response Handler intercepts the response created by backend application. The service endpoint uses a combination of integrated and shared security to evaluate the request before creating the service response. Proposed hybrid approach divides the security tasks between service endpoint's integrated security and the *SAAS Component*. The integrated security at the endpoint performs the primitive tasks, such as encryption/decryption and signing/signature validation etc. The reason why these tasks should be performed locally, is that an endpoint can not accept/release clear text and unsigned messages due to the risk of man-in-the-middle attack, while communicating within and outside the domain. This is done by Security Proxy Handler (SPH), which is an essential part of Handlers. The SPH specifically performs the following security tasks:

1. Encryption/Decryption, Signature/Signature Validation and Key Exchange
2. Enforcement of security decisions, communicated by the *SAAS Component*
3. Report events regarding service requests, responses and enforcements to the logging and monitoring services

Apart from primitive security the other tasks are handled by the security services in the *SAAS component*. The SPH of the service endpoint forwards those tasks to the *SAAS Engine*, which assigns them to the appropriate security service from the SAAS Component. The SAAS Component uses two interfaces for communication with other domains; i.e. 1) *WS-Interface to Security Token Service*, for communication with external identity provider and 2)*WS-Interface for Security Protocols* for execution of security protocols such as Single Sign-On and Non-repudiation with the endpoints of other domains.

6.1 SAAS Component

The *SAAS Component* is a central component, deployed by a security domain for providing shared security to all the service endpoints in that particular domain. Along with various security services, the SAAS component consists of the *Policy Repository* and *PKI Repository*, which are used by different security services. The policy repository consists of the policies, specifying different security requirements, such as *Authentication*, *Authorization* and *Non-repudiation*. The security services in the SAAS Component depend on each other. We discuss below their role and relations in more detail:

1. *Authentication.* The authentication Service provides intra- and inter-domain authentication, as the users requesting services could be from within or outside the domain. In case of an internal request, the authentication service validates user's local identity and sends the signed authentication decision to the endpoint. The Security Proxy Handler (SPH) at the endpoint validates the security service's signature before forwarding the authentication decision to the target service. In case of a request from an outside domain, the authentication service first resolves the identity of the external user. For this, it contacts the external identity provider using *WS Interface* to *Security Token Service* (STS). After the STS validates the user, the authentication service creates a security context for authentication. This provides the functionality of identity federation to the outside user, as discussed in the section 3.
2. *Authorization.* Authorization service verifies permissions assigned to the users. The permissions are defined in the policies stored in *Policy Repository*. Based on the policy, this service makes the authorization decision and sends the signed authorization assertion to the endpoint. The SPH of the endpoint validates the signature of authorization assertion and enforces the decision.
3. *Non-repudiation.* This service executes an out-of-band non-repudiation protocol between requester and the endpoints and stores the protocol messages locally.
4. *Logging.* This service logs the service request and response messages. Messages are communicated to logging service using notifications. Apart from logging, the ordinary requests and responses, the logging service also stores the messages generated by security events. For example, the message notifying that a particular user was not authenticated because the security tokens submitted by her were not signed by the STS. These logs are based on the evaluation done by security compliance service discussed below.
5. *Security Compliance.* This service verifies if the message sent by a user is compliant with the security policy of an endpoint. The security policy defines the security

requirements based on supported security infrastructures and mechanisms, for example, type of tokens, encryption, signature algorithms etc. The authentication service depends upon the decision of the compliance service. If a request is evaluated to be security-compliant by this service then the authentication service proceeds for token validation.

6. *Security Monitoring.* The Monitoring service is responsible to handle the events, which are generated by the endpoints or security services of SAAS Component. For instance, the compliance service reports the security event, if certain message for a service request does not meet an endpoint's security policy. The non-repudiation service notifies a protocol failure, when an endpoint does not follow the Non-repudiation protocol requirements. The monitoring service of a domain's SAAS component notifies these events to a central service, which monitors security among various domains. The purpose of monitoring security centrally is to receive the security events from different domains and notify the responsible and affected endpoints of the particular domains.

From above definitions it follows that security services are not only complex in themselves but also depend on each other. If security is integrated at the endpoint, then it will create very complex security workflows to be handled by the endpoint. Our approach of separating security as a service reduces significant part of security functionality from the endpoints. Moreover, composing all security services as a SAAS component makes it possible to integrate and configure related security components at deployment time.

7 Reference Architecture

The *Reference Architecture* as shown in Figure 3, shows the security services and current web service based security standards for implementing decoupled and shared security. The architecture is divided into three main parts i.e. *Service Consumer* (e.g. a practitioner); *Healthcare System*, which shows different service endpoints in the domain and the *SAAS Component*, which consist of various security services. The *Primitive Security* at the endpoint, performs basic security functionalities of encryption/decryption and signature/signature validation. Its Policy Enforcement Point (PEP) enforces the security decisions at the endpoint. The shared security services shown in the SAAS Component, have already been discussed in Section 6. In the subsequent part of this section , we focus more on the web service based security standards for implementation of the SAAS Component.

1. **WS-SecurityPolicy.** This standard is used to define the security requirements of a service as security assertions [14]. We use this standard to write the security policy of and endpoint, which defines supported type of bindings, tokens, encryption/signature algorithms. In the SAAS Component, this policy is used by two services i.e. security compliance and authentication. The security compliance service checks if the request is according to the policy. Followed by its decision, the authentication service proceeds for token validation.
2. **SAML.** Security Assertion Markup Language (SAML) is used to exchange the security information between security domains [11]. We use SAML for two services

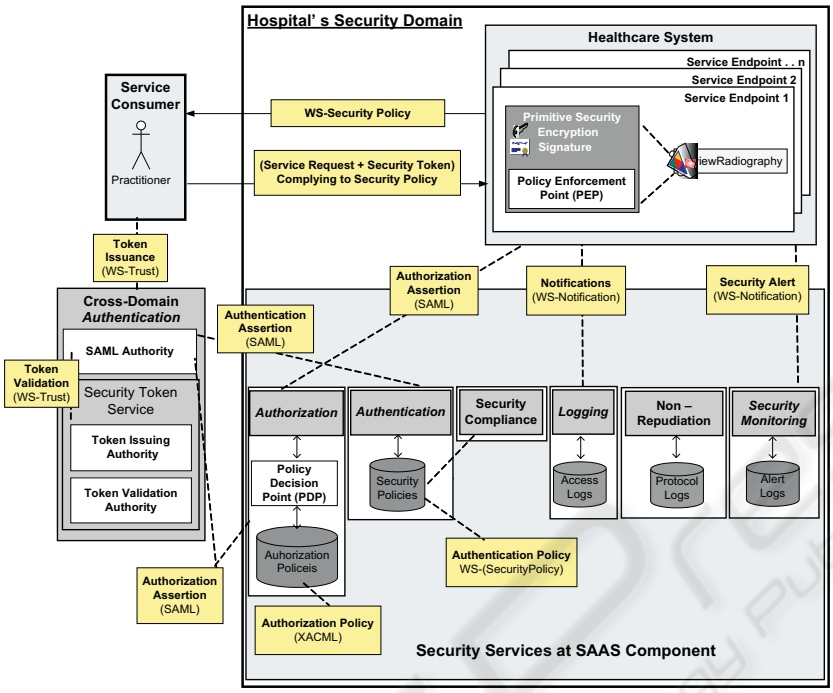

Fig. 3. The Reference Architecture.

i.e. authentication and authorization services. The authentication service creates an authentication request and response based on SAML protocols. The *SAML Authority* gets token validation decision from identity provider i.e. *Security Token Service* (STS) and sends signed authentication assertion to the endpoint. The Authorization service uses SAML in similar manner. It sends authorization request to the Policy Decision Point (PDP) and sends SAML authorization assertions to the endpoint's Policy Enforcement Point (PEP).

3. **XACML.** Extensible Access Control Markup Language (XACML), is a standard to write authorization policies [13]. We use XACML to create authorization policies of a service consumer. *The Policy Decision Point* (PDP) of this service, makes the authorization decision based on the permissions assigned to the roles (e.g. practitioner), defined as XACML rules.
4. **WS-Trust.** WS-Trust provides an interface to service consumer to get security tokens from STS [12]. The authentication service uses WS-trust interface for user's token validation decision from STS.
5. **WS-Notification.** WS-notification defines a set of interfaces to send event notifications [10]. We use this standard to send event notifications from the endpoint to Logging and Security Monitoring Services. Notifications, which are sent for logging carry ordinary information pertaining to the service requests and responses. The notifications, which are sent to the Monitoring Service are the security alerts, resulting from security non-compliance.

8 Composing Security from Components

In Section 2, we mentioned that security services are offered by security components, which are composed based on Service Component Architecture (SCA) model. We use this model to deploy security components as a *Security Composite*, which meets all the security requirements of a domain. An SCA component has a *Service*, a *Reference*, *Wires* and *Properties*. The *Service* is an interface to the functionality a component offers. A *Reference* is an interface of the service that a component depends on. A *service* is connected with a *reference* using a *Wire*. *Properties* are used to configure the components for different implementations. A set of required components is composed as a *Composite*, which provides an interface for service invocation. The composition is recursive and a composite could be used as a component in another composition [3]. Composites are written in XML-based *Service Composition Definition Language* (SCDL). SCA uses SCAPolicyFramework to define security (and other QoS) requirements using high-level security objectives called *Intents*. Intents are mapped to a *PolicySet*, which is mapped to concrete security policies written in WS-Policy standard [16].

8.1 Security Implementation using SCA Model

The motivation of implementing proposed SAAS approach with SCA model is based on its promising features, supporting SOA based composition of security components. We benefit from these features in many aspects as discussed below:

1. **Component-based Security Services.** We assume that the security services deployed in a security domain are offered by components written in different languages. This modularity enables to use old components (reusability), add new components (extensibility) and update existing components (maintainability).
2. **Deployment-time Security Configuration.** Security Components form a *Security Composite*, which can be configured at deployment time based on the security policy of a domain. We use SCA component *properties*, to make such deployment-time configurations. For example, an authentication component is implemented according to the security policy, which defines types of supported tokens (e.g. Signed SAML Tokens, x.509 certificate, Username password), encryption and signature methods (e.g. basic256, sha-1) and message parts (e.g. header, body) to be protected. An authentication component can be written to implement any of these tokens and algorithms etc. The values are passed to the components as properties. This gives the flexibility to configure the same security components for different implementations. In Java, the property-based configurations of SCA components is done with annotations embedded in Java classes [1].
3. **Separation of Concerns.** Using SCA we separate security concerns from functionality, because significant part of the security logic is not integrated with service endpoint. As a result any changes in the security logic can be incorporated without affecting the service endpoint. The service endpoint equally benefits from this separation, because it does not depend upon infrastructure, which are used for its security. Additionally, the peformance of the endpoint is also enhanced as it performs minimum security tasks and does not have to solve interoperability problems.

4. **Dependency Injection.** The security services offered by components depend on each other as discussed in Section. 7. We use *Dependency Injection* to define dependency between security components using SCA *wires* between components.

9 Challenges Related to SAAS Approach

The advantages of the SAAS approach are manifold, but it also offers some implementation-level challenges, which need special attention. SAAS assumes decoupled and centralized security services for each domain. It may create a single point of failure, because many endpoints rely on centralized SAAS Component. This problem should be tackled with certain service replication mechanisms. Secondly, there are only few standards available, which can support SAAS approach, including SAML, XACML, WS-policy, WS-Trust etc. There are no standards for security monitoring, non-repudiation and security compliance at the service and protocol levels. The Web service standards such as WS-Notification, WS-Eventing offer very basic interfaces, which can not be used for complex security scenarios such as failure of mutual authentication or non-repudiation protocol.

10 Conclusions and Future Work

In this paper, we presented an approach for security as a service implementation using Service Component Architecture (SCA) model. We observed that the concept of endpoint security has many obvious limitations in the perspective of SOA. We also investigated that current approaches addressing service-oriented security consider very limited aspects of SOA security. We have presented a more comprehensive set of security services and particularly discussed the aspects of dependencies among security services and deployment-time configuration of security components. In future, we continue our research in two directions i.e. 1) *Security Modeling*, for providing abstractions to the security protocols and 2) *Security Implementation*, for designing protocol engines to implement the security services of SAAS Component.

References

1. SCA Implementation with Java, 2007. http://tuscany.apache.org/.
2. R. Breu, M. Hafner, F. Innerhofer-Oberperfler, and F. Wozak. Model-Driven Security Engineering of Service Oriented Systems. *Lecture Notes in Business Information Processing*, 5(5):59–71, 2008.
3. F. Satoh et. al. Methodology and Tools for End-to-End SOA Security Configurations. In *SERVICES '08*, pages 307–314, Honolulu, HI, 2008.
4. M. Hondo H. Hinton and B. Hutchison. Security Patterns within a Service-Oriented Architecture, 2005.
5. M. Hafner. SECTET A Domain Architecture for Model Driven Security, 2006. PhD Thesis November 2006.
6. R. Kanneganti and P. Chodavarapu. *SOA Security in Action*. Manning Publications Co., Greenwich, CT, USA, 2007.

7. J. Lopez, J. A. Montenegro, and et. al. Specification and Design of Advanced Authentication Authorization Services. *Computer Standards and Interfaces*, 27(5):467–478, 2005.
8. MacAfee. Security as a Service, 2008.
9. M. Memon, M. Hafner, and R. Breu. SECTISSIMO: A Platform-Independent Framework for Security Services. In *ModSec '08: MODELS 2008*, Toulouse, France, 2008.
10. P. Niblett and S. Graham. Events and service-oriented architecture: the OASIS web services notification specifications. *IBM Syst. J.*, 44(4):869–886, 2005.
11. OASIS. Security Assertion Markup Language (SAML), 2005. http://www.oasis-open.org.
12. OASIS. WS-Trust Sepcifications, 2005. http://docs.oasis-open.org/.
13. OASIS. Extensible Access Control Markup Language(XACML), 2006. http://www.oasis-open.org.
14. OASIS. WS-SecurityPolicy, 2007. http://docs.oasis-open.org/.
15. Oracle. Service-Oriented Security: An Application-Centric Look at Identity Management, 2008. http://www.oracle.com/.
16. OSOA. Service Component Architecture, 2007. http://www.osoa.org/.
17. G. Peterson. Service Oriented Security Architecture, 2005. http://www.arctecgroup.net.