Service Oriented Security Architecture

Gunnar Peterson

Arctec Group

"We have no future because our present is too volatile. We have only risk management. The spinning of the given moment's scenarios. Pattern recognition..."

-William Gibson "Pattern Recognition"

Context

New software system paradigms demand that information security keep pace with the changing software landscape. With *Web Services* and *Service Oriented Architectures* (SOA), software development is in the third phase of an evolution that began with object oriented programming and progressed to component based programming. In object oriented and component based programming, security designers could rely on common languages, security models, and technologies in a distributed system to secure both the clients' and servers' transactions and endpoints. For example, EJB clients and servers can assume and use a common J2EE security standard for authentication and authorization for both its client and server. However, in Web Services and Service Oriented Architecture, systems are loosely coupled. Clients and servers may be written in different languages and running on disparate operating systems. Service Oriented Architecture agree upon interface and data schema, but the implementations may vary from client to server, and services may be chained creating a peer to peer model. As programming models and implementations change, the security assumptions and designs must be refreshed and updated to manage the emerging threats that result from new architectures.

Problem Statement

The primary security mechanisms deployed today rely on notions of perimeters and centralized security models. However, the nature of business is moving rapidly towards decentralized "intertwingled-ness" (non-hierarchical connectivity) and perimeters are eroding. Centralized security systems do not apply in SOA's decentralized peer to peer architecture. Security design assumptions based on outmoded technology create brittle and ineffective systems when deployed in a SOA paradigm. Malicious attackers exploit the seams left between the existing security mechanisms deployed based on outmoded assumptions and the reality of the threats to the connected systems on the ground.

Solution

In keeping with the design spirit of SOA, the way forward in software security architecture is viewing security as service that is decoupled and composeable. *Service Oriented Security* (SOS) architecture does not solely focus on perimeters, but rather provides a framework to analyze and manage risk to the system's assets, such as data, services, and identities. SOS and its architectural

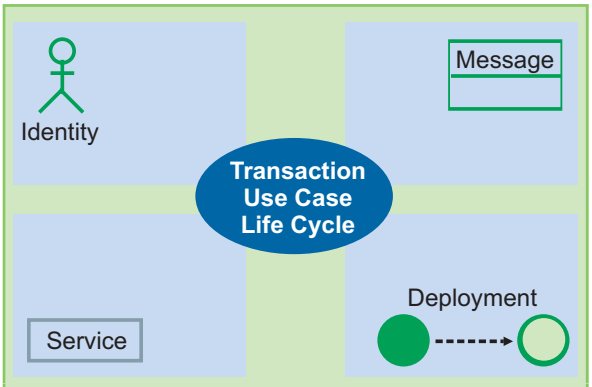

Figure 1 - **Service Oriented Security Views**

views provide a framework for reasoning about software security in a service oriented system. This framework can be used by security architects to harmonize designs across infrastructure, applications, and data based on business risk. Service Oriented Security Architecture provides a set of views that can be used in a larger enterprise security architecture, such as SABSA [7]. In a SABSA model, the five main Service Oriented Security architecture views detail the Conceptual and Logical Security Views.

Service Oriented Security Architecture Overview

In software architecture, the word "security" can often do more harm than good. Frequently, stakeholders have differing, conflicting, and overloaded definitions of the term. In order to build a coherent system, the architects must provide specific guidance to the development and operational teams. Service Oriented Security (SOS) Architecture provides a set of software architecture viewpoints that allow security architects to construct a holistic system design based on a set of views. No single view is sufficient to analyze and understand the system's security as a whole. The combination of the SOS views and their relationships demonstrate the security decisions and design. Since security is not a zero sum game, the views provide a framework in which to conduct security architecture tradeoff analysis and to convey design decisions to development and operational staff. As Kruchten observed [1], views enable the software architect to separate concerns in a complex system. The views in Service Oriented Security consist of the following:

- *Identity View:* deals with the generation, communication, recognition, negotiation, and transformation of identity
- *Service View:* deals with the service, its methods and component parts
- *Message View:* deals with the service's message payload
- *Deployment View:* deals with the defense in depth 5 layer model: *physical, network, host, application, and data*
- *Transaction Use Case Life Cycle View*: deals with the key behavioral flows and relationships in a system and its actors from an end-to-end perspective

Each of the individual views is composed of domain specific elements, constraints, threats, risks, vulnerabilities, and countermeasures. Each view also includes a set of key architectural patterns and principles. By partitioning concerns security designers are able to first decouple concerns to analyze security domains and analyze tradeoffs and dependencies among the domains. The resultant architecture takes the concerns from each domain into account and provides holistic solutions based on the risk management of digital assets like identities and data. The following sections examine each SOS viewpoint in more detail and provide an example usage.

Architecting with the SOS Views

Software security architecture is an iterative process that includes decomposing complex problem spaces, drilling down on granular details to gain traction in a domain; and then synthesizing across domains, building up design views, identifying relationship vectors that illustrate the system's security design goals. Each SOS View contains the following constituents:

- *Elements:* describe the key components in the system and their logical organization including the subjects and objects that the view interacts with
- *Constraints:* show the business, political, legal, and technical constraints for the view
- *Risk Model:* illustrates the threats, assets, vulnerabilities, and countermeasures in the view. Conduct architectural risk analysis to manage risks [8]
- *Relationships:* conveys the nature and direction of the views' relationships with other views

The chief utility of separation of concerns in SOS is that each of the views' elements, constraints, risk models, and relationships are unique and decoupled inside their view. The design decisions made about securing the elements will impact other views, but separating the concerns allows for decoupling the concerns and handling the domain specific risks.

Identity View

Using Kim Cameron's definition we will examine identity as "a set of claims made by one digital subject about itself or another digital subject." [2. This definition rewards careful study because it reveals that identity is not a passive entity, but rather the result of an active set of processes that can be judged against a dynamic set of criteria.

When identity is defined as a set of claims, each service can decide what claims it will accept and from what authorities.

Key elements of the Identity View include:

- Authentication mechanisms, events, and principals including Kerberos tickets, X.509, Windows sessions, and web server sessions
- Identity Federations including the portable, strong identities like SAML, Liberty, and WS-Federation identities
- Monitoring and audit systems: provide traceability of identity-related events like authentication
- Identity domain-specific attributes

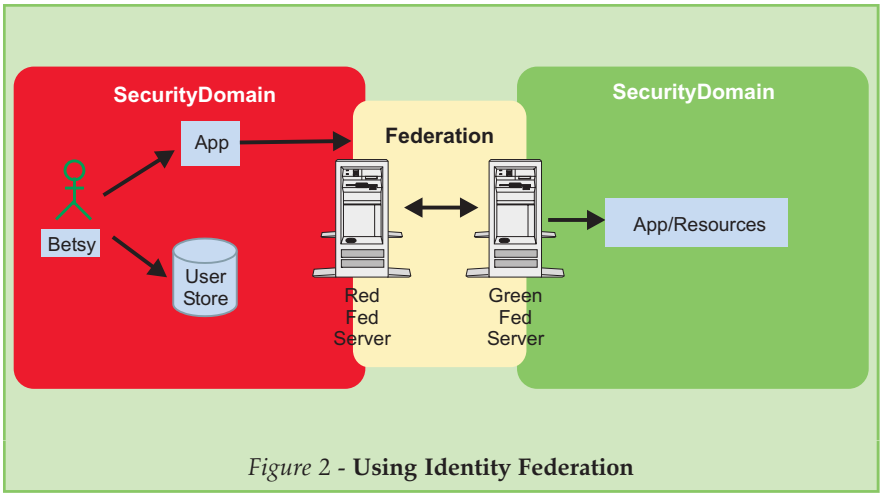

Figure 2 - **Using Identity Federation**

Identity is a fundamental element in access control decisions, hence the protocols that generate, communicate and negotiate identity information require security analysis to ensure a robust system.

Identity information is a very valuable asset to attackers and so systems designers must design its implementation and usage to withstand a variety of attacks. Phishing attacks that continue to grow in distribution and sophistication are just one example of attacks that exploit weak identity systems [3]. Privacy, regulatory, and legal domains all impose constraints on identity implementations and what information may be shared and used by parties. Identity information may "leak" inadvertently to other systems and users that are not allowed to access the information.

Identity View Pattern: Federated Identity

- *Context:* Security domains Red and Green want to integrate disparate systems with unique policies and management. The systems have disparate identity repositories, operating systems, and application languages, but want to exchange XML documents for order processing
- *Problem:* User credentials must be securely ported across domains and security information must be recognizable to both parties
- *Solution:* Use federated identity for access control. Red Client (Alice) logs onto local Red system, generates/sends encrypted SAML token with app request to the Green Service. Green server validates assertions for access rights.

Federation is one example of a service in the identity view. Many other will come into play in a system, including access control mechanisms, anywhere a digital subject is mapped, consumed, stored, and transformed. The Identity View has relationships with the other SOS Views. The Identity View provides information about subjects to the other SOS Views, other views both consume and query principal data provided by the processes generated by Identity View elements. Due to its criticality as a foundation element for other views, the elements in the Identity View should be hardened using the strongest security controls that are reasonably possible for all the states of the system that the identity is in.

Service View

The Service provides the interface to the rest of the system and brokers the identity, messages, and transactional data. The Service View is concerned with the risks around the service and the service's ability to broker information flows with requisite confidentiality, integrity, and availability. Services require access control protection and may consume identity and domain attribute information from other domains, so the service can typically make a minimum of assumptions about what it receives. From a detection and response standpoint, services require logging mechanisms to vouch for the health of the system. Standard technology specific service hardening and security guidelines apply in the services view, such as the OWASP guidelines for Web Applications.

Key elements of the Service View include:

- *Application Servers and Services:* such as databases, web servers, and web services
- *Logging Services*
- *Integrity Services*

Service risks include denial of service, forcing exception conditions, attacks against validation systems, attacks against program and service logic. Since services are an endpoint for incoming requests, they must broker access and data flows between the domains with disparate security models.

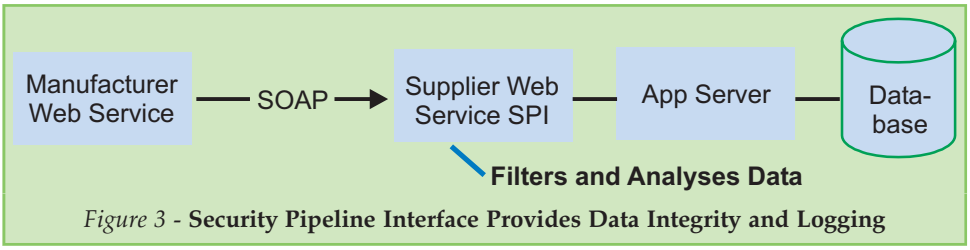

Figure 3 - **Security Pipeline Interface Provides Data Integrity and Logging**

Service View Pattern: Security Pipeline Interface

- *Context:* host must mediate activity between remote client system and back end resources
- *Problem:* host system cannot trust incoming requests and data
- *Solution:* Use *Security Pipeline Interface* (SPI) [4] to enforce the principle of Separation of Privilege [Saltzer75] and reduce risk of data integrity threats. Run SPI in separate physical, process and memory space from business logic aware services. Execute logging of access control and related security event at the SPI.

Message View

Information security is concerned with protecting valuable digital assets, in many cases the most valuable assets from a risk management perspective is not network access, but rather the company's data. As distributed systems continue to evolve and become more connected to each other in ways not foreseen by their original designers, such as decades old legacy systems being connected to the web, data and messages emerge in ways not intended when their protection mechanisms were implemented. The net result of this evolution is to move security mechanisms closer to the asset level, in this case the data elements. Encryption and related technology standards are used to constrain access to persistent data while it is at rest and ensure integrity and auditability over its life cycle.

Key elements in the Message View include:

- *Message Payload:* typically a XML document and related schema information
- *Interface Information:* for example WSDL in Web Services
- *Security Tokens:* such as digital signature information and cryptographic keys

Risks in the Message view include attacks against data at rest and in transit. Since data flows in service oriented systems cannot be predicted in an end to end sense with a high degree of confidence, encryption is a primary consideration for protecting messages as it traverses a host of states, and security domains. As with using PGP for email security, encrypting the message payload provides fine grained protection even without the designer's knowing the precise details of the system that the message will traverse.

Message View Pattern: WS-Security

Context: data is increasingly shared across technological and policy boundaries. The technology stack is unpredictable in SOA systems, hence command and control security systems do not apply.

Problem: message must be protected beyond the span of control of the service and systems, since it can traverse multiple domains. How does the principle of complete mediation [5] apply in a "fire and forget" service oriented world?

Solution: Use WS-Security standard to sign and encrypt persistent XML documents. WS-Security uses XML Encryption and XML Signature, and can accept tokens such as SAML, Kerberos, and X.509 to provide assurance through authentication, authorization, and validation

Deployment Environment View

The Deployment Environment view is focused on the classic information security Defense in Depth layers:

- *Physical:* such as guards and locks
- *Network:* such as firewalls, IDS
- *Host:* such as HIDS, Host Integrity Monitoring and server hardening
- *Application:* such as Input Validation, logging systems
- *Data:* such as database hardening, and data level access controls

The Deployment Environment view is where the lions share of IT security resources are deployed currently. The patterns in place in the Deployment Environment are more mature than those in the emerging SOA-centric views like Identity, Message, and Service views. The risks that are present in the defense in depth layers currently must be baselined against the SOA applications that are deployed on these foundations.

*Deployment Environment View **Anti**-Patterns: Trusted Versus Untrusted Considered Harmful*

- Do not architect using dualistic concepts like "trusted versus untrusted". Deperimeterization renders these definitions meaningless. Instead focus on layers of verification based on available protection, detection, and response mechanisms.

Additional Deployment Environment View Patterns:

- Use Honeypots for understanding the actual threat profile on the ground of each domain to vet trust zone assumptions. Develop metrics

and reporting to feed forward into future security designs.

- Utilize security metrics across layers to develop a scoring system and track performance over time for decision support

Transaction Use Case Life cycle View
Use Cases are used to show the end-to-end view of the system. Use cases provide a synthetic model that correlates requirements from different domains' concerns into a coherent model and flow. Use Case models contain many properties that are critical to secure system design:

- *Stakeholders:* In Information Security, it pays to find allies who have a vested interest in system security. Stakeholders who may be concerned about security implications in the system that is being built include not just the core development staff, but also the legal staff, business owners, domain experts, operational staff, customers, shareholders, and users.
- *Pre and Post Conditions:* Pre and post conditions describe the set conditions that must be satisfied for the Use Case to execute (Pre-conditions) and the set of states that the system can be in after it has completed (Post-conditions). Pre-conditions allow the Information Security team to articulate the security conditions, such as authentication and authorization processes that must be completed before accessing the Use Case functionality so that developers have a consistent spec to build from.
- *Exceptional and Alternate Flows:* A fundamental principle in security design is to design for failure. Development projects are mainly focused on base flows of the system since these implement business valuable features. However from a security standpoint, exceptional and alternate flows highlight paths that often become attack vectors. These flows are worth examination by Information Security to ensure that the system is designed to deal with these exceptions and has deployed security mechanisms such as audit logs and IDS tools to catch security exceptions when they occur.
- *Actors:* Actors can include computer systems, users, and other resources like schedulers. By analyzing the actors involved in the Use Case model, the Information Security team can begin to build a picture of the access control structures such as roles and groups that may be required for design as the system is built. Where delegation or impersonation is used, actors can identify where this is accomplished and what actors are mapped onto other actors at runtime.
- *Relationships:* Much of the power in the Use Case model comes from its simplicity. Use Case models feature two types of relationships: includes and extends. These have direct security implications, in the includes relationship outcome changes the base flow of the Use Case. In the case of including an access control Use Case like Authenticate User, the outcome of this (usually boolean pass/fail) directly changes the behaviors of the related use case. The extends relationship does not alter behavior of the preceding use case, so if a use case extends to a monitor event use case and that monitor server is down, it may not make sense to alter the flow of the preceding Use Case.
- *Mapping Use Cases to Threat Models:* Security cannot only focus on functional requirements, but must also consider the attacker viewpoint. Threat modelling and abuse cases are techniques used in the development life cycle to map possible threats, vulnerabilities, and impacts onto the system so that appropriate security countermeasures can be built into the system. The Use Case model allows the Threat Model to refer to functions in a context-sensitive manner.

Brokering Relationships
To facilitate communication across views and security domains, a common mechanism is needed to generically create, validate, and exchange security information. WS-Trust provides a standard framework for request/response communication for a variety of security tokens including SAML, Kerberos, and X.509. Integration patterns of these relationships can be distilled from the Use Cases. The directionality of the relationship and the nature of the relationships (includes versus extends) shows how the system must be built to satisfy security and privacy requirements across domains.

Architecture Issues

The SOS views describe a way of seeing security architecture across a complex system to make and convey security design decisions. The software security space contains issues that are still being worked to achieve optimal effectiveness.

XML Security
Research has shown various flaws with XML Security [6] related to its reliance on XML for encryption and signature as well as replicating a number of problems in legacy technologies. Since a large number of emerging security solutions, particularly WS-* rely on XML security mechanisms it is worth revisiting this dependency to see if XMPP or other technology can remedy these issues.

Emerging Toolsets and Standards
The software security space is evolving at a rapid pace; investment paths are not clear in a long term sense. Deploying resources based on today's assumptions about standards, for example SAML vs. WS-*, implementation, and threat models inserts a higher degree of variability into

the system's longevity based on the outcomes of the technical and standards challenges.

Changing Threat Landscape: Attacker Co-Evolution

As with any security design, the opponent is homo sapiens meaning that the opponent is ever adaptable and resourceful. As security designs become more robust, then business deploy more resources and transactions to the online world, thus attracting more attackers.

References

[1] Philippe Kruchten, *Architectural Blueprints – The "4+1" View Model of Software Architecture*, IEEE Software, 1995

[2] Kim Cameron, *What is a digital identity?*, 2005

http://www.identityblog.com/2005/03/07.html#a152

[3] Anti-Phishing Working Group, *Phishing Activity Trends Report*, June, 2005

http://antiphishing.org/APWG_Phishing_Activity_Report_Jun_05.pdf

[4] Security Pipeline Interface", Hoffman and Davis, Proceedings of the Sixth Annual Computer Security Applications Conference, 1990

[5] Saltzer and Schroeder, *The Protection of Information in Computer Systems*, Proceedings of the IEEE, 1975)

[6] Peter Gutmann, *Why XML Security is Broken*, 2005

http://www.cs.auckland.ac.nz/~pgut001/pubs/xmlsec.txt

[7] http://www.sabsa-institute.org/

[8] Steven Lavenhar and Gunnar Peterson, *Architectural Risk Analysis*, Cigital Copyright 2005

https://buildsecurityin.us-cert.gov/portal/article/bestpractices/architectural_risk_analysis/architectural_risk_assessment.xml

About the Author

Mr. Peterson is CTO of Arctec Group, a consulting firm focused on building strategic technology blueprints for its clients.

He has served organizations as a Software Architecture consultant for distributed systems including smart card, databases, middleware, messaging, application layer, embedded systems, and commercial packages. Mr. Peterson has provided consulting services on software security design, process, and training to some of the largest financial systems, financial exchanges, manufacturing, and healthcare organizations and systems.

Mr. Peterson has presented on software security topics for numerous conferences and organizations, including ISSA, Black Hat, OWASP, and SANS. He has published papers on the Dept Homeland Security portal: Build Security In on Identity Architecture and on Architectural Risk Assessment. Mr. Peterson is an associate editor of Information Security Bulletin.