

The SOA Magazine

Feature Article



Security in SOA - It's the Car, Not the Garage

by Gunnar Peterson

Published: February 9, 2008 (SOA Magazine Issue XV: February 2008, Copyright © 2008)

[Download this article as a PDF document.](#)

Abstract: Interoperable software architecture requires interoperable security mechanisms. Security is frequently looked at as a black art, but in reality the core concepts of security - knowing your assets and designing for failure - are just good engineering practices. This article focuses on applying those practices to service-oriented solution design with an emphasis on considerations raised by authentication, authorization, auditing, and assurance.

Introduction: SOA Security is About Risk Management

"The revolutionary idea that defines the boundary between modern times and the past is the mastery of risk: the notion that the future is more than a whim of the Gods and that men and women are not passive before nature. Until human beings discovered a way across that boundary, the future was a mirror of the past or the murky domain of oracles and soothsayers who held monopoly over knowledge of anticipated events..."

- Peter Bernstein, "Against the Gods"

When I park my car in the garage, I lock it. Why? Well, although I would hate for someone to steal my snow shovel and hockey sticks, my car is much more valuable to me. Security is about managing risk, specifically protecting valuable assets like my car. I have a higher level of protection on my car than on my garage. In dollar terms, the contents of my garage are orders of magnitude less valuable than my car. I could spend a lot of money fortifying my garage, and that would add some security to my car while it is parked there, but it is not a cost-effective investment. First, my car is the asset of value, and second the garage - no matter how well protected it is - doesn't move.

Car manufacturers know this, insurance companies know this, consumers know this. Even media publishers know, yet in the common enterprise, programmers and architects seem to roam in ignorance. Your average download of a Michael Bolton song carries a far higher level of security than valuable user data, like passwords, social security numbers, and credit card details. Why do we keep protecting critical data with point-to-point security solutions (like SSL) that protect the transmission channel, but leave the valuable assets being transported wide open everywhere else? This is a critical question that needs to be answered in order to successfully add an effective layer of security to an SOA.

218 Million Breached Records Can't be Wrong

The stats are in, and they are not pretty. Data breaches of sensitive consumer and enterprise data occur on a regular basis. In the U.S. alone, there have been over 218 million data records breached since 2005 [REF-1]. We are approaching a "dog bites man" level of regularity here, except you'll need more than a rabies shot to recover. When your sensitive data bites you (via, say, an identity theft) it is likely going to be expensive and time consuming.

Security Services

Each progression in distributed computing - from object-orientation to component-based design and now to service-

orientation - has introduced unique security considerations. Objects and components use similar binary runtimes, but when building services as Web services, we can no longer rely on binary controls for security (Table 1).

	Object-Orientation	Component-Based Design	Service-Orientation with Web Services
Paradigm	abstract models (objects) used to bundle data and methods	technology-specific implementation model for distributed programming	services designed as autonomous and standardized programs with an emphasis on reuse
Security Implications	vendor or implementation provides security mechanisms, authentication, authorization, audit, for object processing runtime (example: Java authentication and authorization services in JDK)	component model provides implementation specific security models (example: container security in EJB)	interoperable industry security standards deal with message security, identity federation, and service security (example: WS-Security)

Table 1: Different distributed programming paradigms introduce different security considerations.

The primary security functions required by most systems are:

- authentication
- authorization
- auditing
- assurance

In SOA, these four functions can correspond to processing functionality provided by reusable utility services, which I'll refer to as "security services." Figure 1 shows how security services are needed to mediate communication between a subject and its objects - or, in the SOA world, between the service provider and its requesters (or consumers).

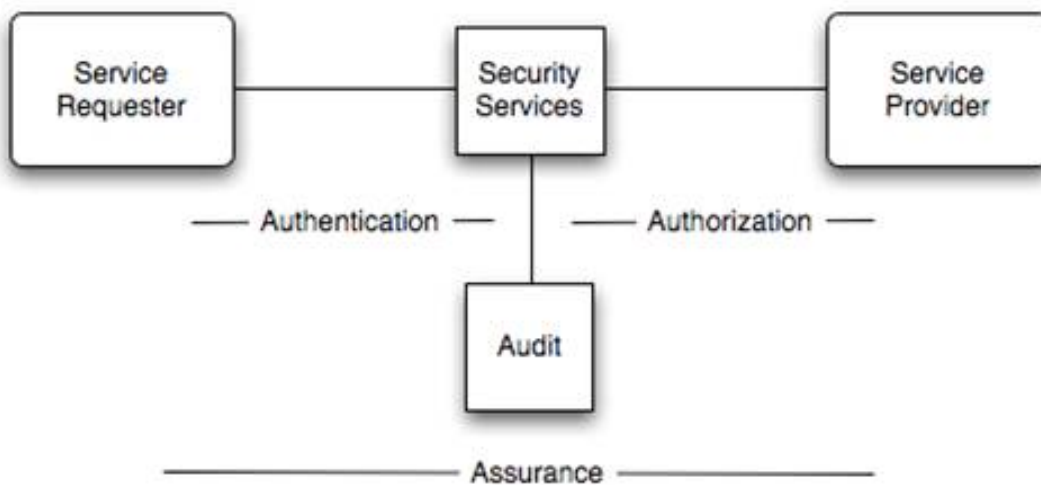


Figure 1: How security services can be positioned as intermediaries between service requester and provider.

A base SOA use case consists of service requesters, service providers, and message exchange patterns. It is within the message exchange where the authentication, authorization, audit, and assurance services add true value.

From a development perspective, the authentication and authorization services are frequently combined at runtime, but when classifying them as different service models, a logically decoupled service layer view is established.

Let's take a closer look at each of the security-centric service models:

- Authentication is concerned with validating the authenticity of the request and binding the results to a principle. This is frequently a system-level service because it deals with the processing of system policies (such as password policies) and implementing complex protocols (like Kerberos). This warrants a separate service because authentication logic is generally not valuable (or reusable) when intertwined with other application logic.
- Authorization, on some level, is always enforced locally, close to the thing being protected. In SOA, this thing is the service provider. While coarse-grained authorization can be implemented at a global level, finer grained authorization requires mapping to the service and its operations. From a design perspective, authorization should be viewed at both system and service levels (the latter always being enforced locally).
- Audit services provide detection and response features that serve to answer questions around what digital subjects performed what actions to what objects.
- Assurance services essentially exist as a set of system processes that increase the assessor's confidence in a given system.

Standards First

SOA security architects have been blessed with security standards popularized by the SOA movement and should therefore use them to the greatest extent possible. These standards are subject to the rigor of committee reviews and therefore benefit from a higher level of quality assurance than typical custom solutions.

That said, we do not yet have a completely standardized set of security services available. We do have standards for authentication, some interesting early stage standards in authorization, and nothing to speak of yet in auditing. It is now well understood in the security community that solutions should be built with standard cryptographic libraries. Luckily .Net, Java and other programming platforms provide implementations that give developers access to RSA, SHA-256 and other crypto algorithms.

This still leaves us with at least two major problems:

- encrypting and decrypting (it's nice and all, but we still need to move that data around and use it)
- access control

Fortunately, the SOA movement has produced useful standards like WS-Security that help solve the first problem (moving data around). WS-Security SOAP headers facilitate encrypting data in the message and because the data is packaged in XML, other service providers can decrypt the message. Additionally, WS-Security allows for multiple security token types, so if your enterprise is using Active Directory, LDAP, and digital certificates, you can still mesh security requests together in a consistent manner.

The second issue of access control is the focus of the remainder of this article. Access control is comprised of authentication (who made this request?), authorization (what is this request allowed to do?), and auditing (what security decisions were made for what requests?).

The Word "Security" Considered Harmful

From years of software security consulting and training I can tell you one thing for sure - "security" is an overloaded and misused term and one that should be retired as soon as possible. One of the first things I do on a project is to ask developers and architects to cease and desist using the word "security."

Ask five different people to define "security" and you are likely to get six different responses. Most will say "but we have a firewall...", some will respond with security policy, others with access management, others with crypto, and so on. These elements may all be part of a security solution, but they do not "secure" much of anything in a holistic sense. Lack of clear agreement over the definition of security leads to incorrect assumptions about who is doing what, further leading to solvable problems not getting addressed due to low resolution communication. All this (and 218 million breached records) from one word.

Instead, when discussing security, try to be more specific. Use terms like "confidentiality," "integrity," and "availability"; as in: "we need integrity services for this message that is going over the wire because we want the service provider to know they received a complete message", and then drill down from there. For example, avoid saying "we need to secure the message." (What does that even mean?) Try terms like "authentication," "authorization," and "attribution"; as in: "we need to delegate authentication from the Web

Authentication

I can generally empathize with developers that push back against some security constraints that might seem like overkill. But one that I have never understood is digital signing. Why on earth would you want to make business decisions based on an unsigned message? Forget about confidentiality and data breaches for a moment, and think of data integrity. Unless you have some way of guaranteeing that you have a complete record, how do you know your system is making good decisions?

Identity is the basis of all access control decisions, and most of these decisions generally begin with authenticating the request.

However, here is the wrinkle for SOA security architects: it is highly likely that the enterprise already has numerous directory services and that, for any given application, already authenticate the user client. So the first question becomes: how does the system reuse the information from the authentication events that have occurred?

Tip: Federate, Don't (Re)authenticate!

Federated identity has emerged as the rarest of things - a scalable, security solution that does not detract from usability. Establishing federation between policy decision points within the service requester and service provider enables the request to carry the highest degree of security.

Authorization

Authentication is an inherently forward facing activity, focused on the request. Authorization, on the other hand, is a set of resource facing decisions, that ultimately grant or deny access to something. Vendors and standards communities have been heavily focused on writing and implementing authentication standards, but authorization standards remain behind the curve for now.

There are some promising efforts in the authorization field that naturally build upon authentication standards such as XACML (a policy language that can use SAML assertions). XACML, in fact, is a good example of the problems that an authorization scheme must solve. In XACML, a policy enforcement point (PEP) intercepts a service request, gathers all relevant information and credentials about the request, and then passes them to a policy decision point (PDP) to render a decision on granting access. The basis of the decision is a three part rule where the resource that is attempting to be accessed is mapped to a condition and an action for a subject.

So a resource (for example a URL like <http://example.com/creditcard>) is mapped to a condition for access based on the action (like "read only") the access is granted or denied to the subject (like a user or group). Pretty standard stuff, but what's so special about XACML is that it enables this authorization logic to be widely reused in the enterprise. Think of how many places a service-oriented application can touch security policy logic (directory, operating systems, databases, firewalls, routers, and so on).

Tip: Validate "Who goes there?"

Before granting or denying access, actually before you even start the process of asking the question of the PDP, you must always validate the data. Services are loosely coupled, which is great for interoperability and scalability but also great for attackers to inject nefarious information that targets the host, data and other resources. This has given rise to SQL injection, XPath injection, and a whole array of other attacks.

The very first line of defense is to validate the message data. This is done against a whitelist (default deny) which specifies the data sets, types, and values that are allowed to be used; and/or a blacklist (default allow) which specifies a list of known bad data sets, types, and values. The important thing is that all data is guilty until proven innocent. There are simply too many ways for an attacker to inject data. But be forewarned, in building whitelists and blacklists, security people will be greeted as warmly as TSA workers confiscating toothpaste and having people remove shoes. Good times.

server to the app server", and then drill down from there on authentication types, tokens, policies, and so on.

For example, instead of saying "we need to secure the session" (how are you going to "secure" a session? ...run it on the NSA's campus?), use terms like "input validation"; as in "all input is validated upon entry to server", and then drill down from there on input validation: whitelist, black list, and so on.

Auditing

Security is a three part process: protect, detect, and respond. So far we have discussed authentication and authorization protection schemes. Let's turn our attention to auditing, which will help us understand what security decisions have been made and therefore makes for an important part of the response process.

Tip: Roll Your Own Auditing

I mentioned earlier that standards are important, but if you don't have any what do you do? In the case of auditing, the vendors and standards bodies have let us down. All we can really do at this stage is create an audit logging solution from scratch. For example, we could bend some non-security standards to our will, as shown in the WS-Addressing example below.

```
<soap:Envelope...>
  <soap:Header>
    <wsa:To>http://localhost/MyService</wsa:To>
    <wsa:ReplyTo>
      <wsa:Address>http://localhost/MyRequester</wsa:Address>
      <wsa:Address>http://localhost/MyAuditLog</wsa:Address>
    </wsa:ReplyTo>
    ...
```

Example 1 - A custom WS-Addressing header containing security details. Adding the Audit Log to `wsa:ReplyTo` notifies the service to send a carbon copy of the response to the audit log, which can then be used for audit reporting purposes.

Note: Be sure to look to XDAS [REF-2] for some lessons learned on distributed audit logging so you don't reinvent the wheel.

Assurance

Assurance is a large topic area, but I would like to focus on two cost-effective assurance activities. First, static analysis solutions are beginning to address Web services, XML and related technologies. This means there are tools available to scan your code while it is still under development to identify security-related programming bugs. (I cannot imagine not scanning code on every development project.) Secondly, black box is a proven means of scanning the service and host environments to identify known vulnerabilities and bad configurations.

As with static analysis and blackbox scanning tools, white box code scanning tools allow developers to find security bugs and not repeat the mistakes of the past. You can use these tools to cost effectively identify known vulnerabilities in your system before it goes live.

Putting it all Together

Here are some action items to help you incorporate security into SOA:

- Map out a security architecture that looks at the system from an end to end perspective, and focuses on your assets (it's the car, not the garage).
- For each service requester and service provider - and anything in the middle like proxies - understand the current state of access control by analyzing authentication, authorization, and auditing (secure access to the car).
- Determine what policy enforcement and policy decision points exist today and which can be strengthened in the future (fortify the car to the best of your ability).

Security is not a goal in and of itself. It is a business enabler. The great Robert Garigue [REF-3] said that security is like brakes on a car. Because we have brakes we can drive faster. SOA security architects, this is your mantra. Find ways that you can deliver security services to your organization while enabling your business to grow.

In general, security primitives are available and in some cases optimized for enterprise use. However, being able to

design the actual mechanisms of a service-oriented system remains a critical success factor for SOA. It is important to acknowledge that security in SOA is different from infrastructure-related security. While the latter is focused on establishing perimeter defenses like firewall moats, SOA security is very much concerned with what the system is supposed to do and what can go wrong.

In many cases, the most important thing the SOA security architects brings to the table is designing for failure. Whether the failure is generated on purpose or through malice is not as important as having a resilient system that spans technologies and domains and enables the business to complete its mission.

Conclusion

There is no perfect security solution, there is only the management of security risk that relies on judgment and prioritization, driven by assets and values. Security is contextual and has a form factor that must adhere to that which the supporting mechanisms can protect. Risk is increasingly engendered in data and effective security mechanisms adhere to data to provide the necessary level of protection. When SOA security standards are properly leveraged, the potential is there to create entirely new and robust service-oriented security architectures.

References

[REF-1] "A Chronology of Data Breaches", Privacy Rights Clearinghouse, <http://www.privacyrights.org/ar/ChronDataBreaches.htm>

[REF-2] "Distributed Audit Service (XDAS)", The Open Group, <http://www.opengroup.org/pubs/catalog/p441.htm>

[REF-3] "Thinking about Robert Garigue", 1 Raindrop Blog, http://1raindrop.typepad.com/1_raindrop/2007/02/thinking_about_.html

THE PRENTICE HALL SERVICE-ORIENTED COMPUTING SERIES FROM THOMAS ERL



[Home](#) [Past Issues](#) [Contributors](#) [What is SOA?](#) [SOA Glossary](#) [SOA School](#) [SOA Books](#) [About](#) [Legal](#)

Copyright © 2006-2008
SOA Systems Inc.