# Software Security and SOA: Danger, Will Robinson!

Jeremy Epstein
*webMethods*

Scott Matsumoto and Gary McGraw
*Cigital*

The current buzzword of choice among the technical elite (at least those subject to marketing departments) is *service-oriented architecture*, or SOA (pronounced "SO-uh"). As SOA moves from hype to practice, an opportunity exists to do security right, but a similar opportunity exists for disaster if security is done wrong. This article describes 13 snares that we must avoid to end up with SOA security that makes sense.

## SOA what?

Getting to the bottom of exactly what SOA is might seem a bit daunting, especially given the acronym-laden, hype-induced hyperbole floating around the Internet. The buzz about SOA today reminds us of the buzz about Java in the mid 1990s—only somehow even more vague. Will it really clean everything up and send it off to college? The real question is whether there's even a bottom to get to. It seems as if SOA amounts to a Web services-based architecture that relies on a data-driven XML model. The notion is to provide various services over an enterprise bus that many diverse applications can access and use remotely. As you might imagine, doing this right involves knowing plenty about how the business actually works and thinking clearly about what architectural components make sense.

An enterprise information architecture that's aligned with business is a good thing, and to the extent that SOA's adoption helps with this alignment, the SOA itself will be useful. However, past architectural paradigms have had to deal with three common problems:

- inability to create composite applications nimbly (with parts borrowed from multiple large enterprise applications),
- difficulty in defining clear services based on encapsulating myriad diverse APIs from large enterprise applications, and
- problems in containing and managing services in a centrally reasonable way, leading to the proliferation of closely related but difficult-to-manage clones.

SOA promises to address these three problems in one fell swoop, by defining language- and hardware-independent protocols and representations for loose coupling of software components.

Ultimately, a Web services-based SOA also seems to be a relevant target for many enterprises. The difference between a random collection of Web applications and a SOA approach lies mainly in the use of a common XML-based data model and an intelligent transport/transmission bus. Think of moving from an RPC-centric model to a document-centric model to grok the difference: message granularity shifts from specific calls to APIs toward larger globs of data+state messaging built out of XML. In some cases, the transmission bus itself does all the necessary data restructuring to allow different services to communicate.

## What about security?

It almost goes without saying that large enterprises are obsessed with security and securing their critical applications—their essential data are at stake. Any move toward SOA presents a prime opportunity to build security into future applications.

But with every opportunity comes a danger of seriously screwing things up. Early SOA adopters are already falling prey to bad thinking about security. The biggest problem by far involves confusing software security with security software (note the word order). As this department emphasizes, security isn't a feature. Just as you can't sprinkle crypto fairy dust on software to make it magically secure, you can't liberally apply crypto to SOA and end up with security. Crypto is security software; what we want is software security.

Those considering SOA would do well to give close consideration to the inherent security of the Web services platform, as well as to the services themselves. SOA presents an opportunity to avoid or otherwise manage security flaws that pervade software architecture (accounting for 50 percent of the software security problem).[1]

In the messy real world of enterprise applications, vendors typically highlight the primary security features that they offer as a key selling point. However, outside the list of mandatory security features, few

vendors can attest to the underlying security of the product itself. Thus, users might have all the security features in the world, but they remain untenably insecure.

The challenges of insecure software grow with the move to SOA, which by its very definition exposes software vulnerabilities more widely than ever. Heck, given the Web Services Description Language (WSDL) and universal description, discovery, and integration (UDDI), gaining the information you need as a malicious attacker to perform a software exploit is now easier than ever. In this scenario, a variety of standards might take the place of arbitrary and often-broken home-grown security features, but the results remain the same because the services themselves suffer from shoddy construction.

By understanding the common snares we list in the next section, those enterprises considering SOA technology can ask better questions, such as, "How do I know that SOA product 57 is secure?" and "What kinds of measures have been taken to avoid software security defects?"

## Thirteen security snares

Without further ado, let's get into that list:

- *Assuming the vendor will take care of security*. When you buy a new car, you don't ask about the engineering processes used in the design; you assume that Ford or Toyota knows more about how to design cars than you do. Of course, government oversight helps with car safety, but because there is no such oversight on software vendors today, you can't assume that vendors will take care of it. They have a propensity to "check off the security box" by throwing in some crypto features and calling it a day. Even SOA security vendors such as Vordel and Reactivity are focusing their attention on reactive approaches instead of telling people they need to build security in.
- *Not asking about security at all*. Many IT organizations (even in large companies) have no dedicated internal security staff. Even in organizations with great network security staff, little or no attention is paid to software security. Because SOA is about software architecture, security might not be something that even comes up.
- *Asking about the wrong kinds of security things*. On one hand, IT security personnel are likely to believe in the religion of the firewall; in fact, SOA has already engendered its own firewall sect. However, reactive approaches haven't worked out very well for security, and they're not likely to start working soon. On the other hand, software people are likely to fall square into the "security software" hole. As we said earlier, security features alone don't make for software security. Building secure software means applying the software security touchpoints[1] and thinking about security during design and implementation.
- *Allowing discomfort with the technology to overcome the need for software security*. Most network security engineers feel comfortable with the bits and bytes of routers, firewalls, and operating systems, but few know much about the security of enterprise business applications or the SOA itself. As a result, they tend to ask about the aspects they're familiar with—such as use of SSL—and ignore the harder questions like, "How can you demonstrate to us that this product is secure?" Getting outside your technology comfort zone is often elucidating and educational. Do it.
- *Relying on a cursory risk assessment*. Smart organizations know how to manage risks, and they make conscious decisions about where to focus their limited resources. Some believe that even if their SOA framework has security

flaws, the odds of those problems being detected and exploited is far lower than the odds of an attack through an unpatched router or Web server. Today, evil people are attacking commodity network products more often than customer-specific Web services, but this is quickly changing. Software security is becoming increasingly important because attackers are changing their targets—risk assessments shift over time to account for an attack's probability for a reason. Whatever you do, don't forget about software security and software flaws during risk assessments.

- *Believing you're secure for no apparent reason or for the wrong reasons*. Everyone's heard that old chestnut, "We're safe because we're behind the firewall," and in too many organizations, this is treated as gospel. Because Web services tend to be implemented on servers inside the organization, worry about their inherent security is often discounted, but the fact remains that most firewalls will simply pass along a Web services request, including any attack code. SOAP is the attacker's friend, and the age of the firewall is drawing to a close. The very idea of SOA is to promote reuse and repurposing of common functionality; it's about externalizing the data schema and thereby further erasing the distinction between "inside" and "outside."
- *Misapplying vulnerability metrics*. It's easy enough to search databases of vulnerabilities (such as the CVE or BugTraq) to see how many security problems have turned up in a given product and how severe they are. Rather than asking the vendor directly about security, some security engineers incorrectly rely on public metrics such as the number and severity of publicly reported bugs to determine the product's quality. Whether these metrics are correlated with actual product security remains an open research question. A better idea is to ask a vendor questions about security assurance in their SDLC: do they use the touchpoints?[1]
- *Trusting the vendors (too much)*. Vendors might intentionally or unintentionally give inaccurate results. A vendor who performs penetration testing, for example, might not have tested the product or version being considered, thus the testing's value might be reduced. But some data are more useful than others. Does the vendor review code with source-code analysis tools such as those from Fortify, Secure Software, or Ounce Labs? What kinds of results can the vendor show you?
- *Building a proof of concept that ignores security "for now."* SOA gurus recommend the "start small and grow" approach to re-architecture. This approach most often results in a proof-of-concept application that acts as a test of some framework or collection of vendor products. Frequently, security isn't a requirement in this proof of concept, and technical issues other than the functional success of the proof of concept aren't revisited before a procurement decision is made. Thus, the opportunity to consider software security is missed. Don't leave security for later—ever. Ironically, the "security as a feature" approach at least opens the door to a discussion of security.
- *Believing security is somebody else's job*. This could be a variant of "assuming that the vendor will take care of security," or it could be a symptom of an organization in which security specialists aren't responsible for the security of the development systems in use. Software security is everybody's job. By involving development in security, we cut through the myth of security by firewall.
- *Giving up hope*. The security specialist (if there is one) knows that he or she can only say "no" so many times and only has a limited amount of influence over purchasing decisions. Why spend the time questioning a vendor or analyzing a SOA platform's security when his or her actions are unlikely to impact the procurement or deployment decision? For this person, it seems easier and better to use any silver bullets to influence a critical piece of security infrastructure. Don't give up hope. The only way we'll fix the security problem we're in is by building better software.
- *Putting too much weight on security standards and security features*. Standards such as SSL (for Web servers), S/MIME (for email), and WS-Security (in the Web services space) are widely perceived to provide security. Too many organizations fail to understand that although these standards are important, they don't actually do anything to secure a system. An implementation bug or an architectural flaw in a product can leave a system that's completely standards-compliant completely insecure as well. Use a discussion of security features as a flying wedge to talk about more intensive software security assurance.

**Everyone's heard that old chestnut, 'We're safe because we're behind the firewall,' and in too many organizations, this is treated as gospel.**

- *Doing it all yourself.* To end this list on a positive note, some organizations don't ask the security question because they plan to come to their own conclusions by performing their own hard-core analysis and testing. Good!

Despite the failure of users to ask about software security when it comes to SOA, vendors are actually quite willing, able, and, in many cases, eager to provide improved security in their products. In other words, there's adequate supply. The problem is that there's insufficient demand, at least as expressed in buying decisions.

## All for SOA, SOA for all

Why don't we ask the security question of every software vendor we interview (SOA or otherwise)? In many cases, the decision to do so could be entirely reasonable. Whenever we look at a vendor, we should make an explicit decision whether the product's security is important, and if not, document why it's not. For many organizations, our 13 SOA security snares could be the right starting point.

For those vendors from whom we want and in fact need to know how secure the product is, we need to know what to ask and how to assess the answers. Among the measures a software vendor might use to increase the robustness of their products against security failures, you should look for some of the following:

- strong security involvement in architecture or design,
- good software engineering practices,
- security-focused quality assurance (QA),
- penetration testing,
- automated vulnerability testing,
- manual or automated source code analysis,
- defect density prediction,
- developers trained in software security,
- a development methodology, such as the touchpoints, that helps identify security problems before they occur,[1] and
- other third-party reviews.

Unfortunately, there's no single answer to how much is enough. Should vendors be expected to meet all or most of them? How do we prioritize between competing claims? How should we compare two months of security-focused QA, for example, to a week of automatic code analysis? Is a product that has undergone a BITS (www.bitsinfo.org/about.html) evaluation more secure than one in which all developers are trained on software security touchpoints?

In reality, these questions aren't that different from those raised in the procurement cycle, such as the trade-off between cost and performance, with one exception: these are the criteria that aren't typically assessed in a formal manner as part of the process. It's important to remember that no evaluation process guarantees a product's success, but it does help improve the odds of success while providing us with additional recourse should issues arise. Following a proof-of-concept exercise, for example, we can feel relatively certain that we've identified reasonable performance. By extending a product's underlying security to similar scrutiny, we can not only improve the likelihood that a vendor product won't expose a security breach within the enterprise, but also provide greater insight into its security architecture so that we can more readily bolster any uncovered shortcomings.

For organizations building SOA via Web services, solving the security problem requires both a secure SOA framework and securing the Web services themselves. Purchasers of Web services platforms can and should ask their vendors about how they secure their platforms. And developers of Web services on top of these platforms should take an equal responsibility and introduce rigor in their design and testing, so that the resulting Web services don't become the "hack me" locations of choice.

By asking SOA vendors, "How do you know your product is secure?" organizations will raise the bar for software security. ▢

## Acknowledgments

*We thank John Steven of Cigital for his insights on SOA security and the 13 SOA security snags. An earlier version of this article appeared as "Software Security Supply in Demand," Web Services J., vol. 5, no. 11, 2005, pp. 26–27; http://pdf.sys-con.com/WebServices/WSJNov2005.pdf.*

### Reference

1. G. McGraw, *Software Security: Building Security In*, Addison-Wesley, 2006.

***Jeremy Epstein*** *is a senior director of product security at webMethods, a provider of business integration and Web services software. His research interests include methods for improving software quality in commercial software and security for e-voting systems. Epstein has a BS in computer science from New Mexico Tech and an MS in computer sciences from Purdue University. He is a senior member of the IEEE and a member of Usenix. Contact him at jeremy.epstein@webMethods.com.*

***Scott Matsumoto*** *is a principal consultant at Cigital in the enterprise architecture practice. His research interests include DSLs and development tools. Matsumoto has a BA in physics from Lawrence University. Contact him at smatsumoto@cigital.com.*

***Gary McGraw*** *is chief technology officer of Cigital. His real-world experience is grounded in years of consulting with major corporations and software producers. McGraw is the author of* Software Security *(Addison-Wesley, 2006), and coauthor of* Exploiting Software *(Addison-Wesley, 2004),* Building Secure Software *(Addison-Wesley, 2001),* Java Security *(John Wiley & Sons, 1996), and four other books. He has a BA in philosophy from the University of Virginia and a dual PhD in computer science and cognitive science from Indiana University. He's also a member of the Board of Governors of the IEEE Computer Society. Contact him at gem@cigital.com.*