Building Security In
Editors: John Steven, jsteven@cigital.com
Gunnar Peterson, gunnar@arctecgroup.net
Deborah A. Frincke, deborah.frincke@pnl.gov

Logging in the Age of Web Services

Anton Chuvakin
Qualys

Gunnar Peterson
Arctec Group

In today's age of Web applications connected via Web services, accountability has become both crucial and harder to achieve. The management of authentication, authorization, and accountability in these applications is therefore a very important and difficult problem to solve. In this article, we describe how audit logging can be built into the Web services infrastructure.

Why Web Services?

Web services aim to deliver virtualization, interoperability, and reusability via technology implementations such as SOAP, service-oriented architecture (SOA), and Representational State Transfer (REST). Interoperability in particular is paramount for enterprises that must transact business across multiple .NET, Java, open source, and mainframe platforms. Add in messaging connections to customers and partners located in different countries or organizations, and the challenges of getting protocols and message formats to mesh together seem overwhelming. This is the problem that Web services attempt to address. Unfortunately, however, most Web services implementations offer their own challenges as well—namely, they don't always enable security by default, and they often leave security decisions about authentication, authorization, and audit logging to the implementer, which can cause serious long-term problems down the road.[1]

Web services security has two parts: interface and implementation security and message security. Interface and implementation security uses traditional Web application security controls such as Secure Sockets Layer (SSL) and access control lists (ACLs). For message security, XML mechanisms such as WS-Security, the Security Assertion Markup Language (SAML), XML Signature, and XML Encryption can sign, encrypt, and authenticate message data, giving the sender greater confidence that the message stays confidential until delivery and the recipient greater confidence in the message's authenticity.

Instead of relying on role-based access control, in which security services mediate the communications of subjects, objects, and sessions in a central policy domain, Web services evaluate a message's claims against a specific policy. This claims-based access control model lets the architect compose security protocols along the same lines as the application. Standards-based XML security mechanisms such as WS-Security, SAML, XML Signature, and XML Encryption are powerful tools for people building security architectures in distributed systems to deliver message-level authentication, integrity, and encryption services. These standards offer new and more robust ways to protect Web services, but they don't provide services for logging, audit logging, and detection and response. Moreover, Web services architectures are generally built on stateless protocols such as HTTP and SOAP, which don't support even basic request-response correlation, leading to a major security architecture and design challenge.

Logging and Web Services

Logs offer an endless well of valuable information about systems, networks, and applications. Through logs, audit records, and alerts, information systems often give signs that something is broken (or "broken into") or will be broken soon. They can also reveal larger weaknesses that might affect regulatory compliance and even corporate governance. However, more often than not, system and application logs contain raw data rather than information, and thus require extra effort to extract or distill this data into something useful, usable, and actionable.[2]

At the very highest level, logs are a vehicle of accountability. Of course, an organization has many other mechanisms for accountability, but logs pervade all of its IT, and if its IT isn't accountable, the organization probably isn't either. Various logs are also valuable for regulatory compliance programs. Many recent laws and mandates have items related to audit logs—for example, a detailed analysis of the security requirements and specifications outlined in the Health Insurance Portability and

Accountability Act (HIPAA) reveals some items relevant to auditing and logging, such as the audit, logging, and monitoring controls for systems that contain patient information. Similarly, the Payment Card Industry Data Security Standard (PCI DSS) explicitly mentions logging, log collection, and data retention and log analysis for systems involved in credit-card transactions. Clearly, the importance of logs for compliance will only grow as standards become the foundations for new regulations that are sure to emerge.

An interesting challenge that's likewise sure to emerge is how to handle logging in distributed environments such as those of Web services. Unlike operating system logs, which are physically located on a single machine or a single network device, Web services are by their nature distributed across multiple systems, disparate technologies and policies, and even organizational domains. This creates many constraints on audit log design in terms of both scope and strength. Given these constraints, logging might reside in different parts of Web services architectures:

- In the simplest case (two machines talking to each other via a Web service), we could probably log events occurring on both machines, but we won't have a complete view of the overall architecture. If we put a log on each machine, we'll need additional technologies for consistent aggregation and reconciliation.
- At each endpoint, Web services interaction involves multiple software components. At the very least, this will involve a Web server, an application server, and most likely a database server on each side. Moreover, the application server might be self-distributed, thus presenting additional challenges to determine where to put the log.
- Even under a distributed Web services architecture, we can choose an approach in which each individual component on each individual machine creates its own log. It's not uncommon to see applications log into their own text files; similarly, it's not uncommon for Web services designers to use the MS Windows event log if their endpoint applications only run on this platform. (Many application developers tend to choose the Windows event log because they consider it to be the standard for logging.)

Reviewing these options leads us to believe that—where practical—a Web services architecture should have its own logging mechanism, so that each component involved in that particular architecture has a uniform way of creating a log for later access from a single point. Moreover, each component in such a distributed system should not only use a single mechanism but also use the same event types to let the user reconstruct what actually transpired in the infrastructure. It's worth noting that some of the events to log might be distributed in nature themselves, which could prevent anyone from using a local log to describe them. Therefore, the best way to perform logging in a Web services architecture might be a fully distributed logging mechanism. Despite its distributed nature, application server logging is probably the central part of Web services logging; Web server logs are less helpful because many Web services (especially SOAP), servers, and even application firewalls specifically turn off logging for HTTP POST tasks because of the amount of data generated in these requests.

Classic Web services have a clean separation of the message document (typically XML) and the implementation (Java, C#, and so on), so logging location varies based on intent. The Web services framework often implements logging events based on message content in an adapter, handler, or interceptor type pattern. This approach's advantage is that it executes logging rules after message creation but before the sender puts it on the network and as soon as the recipient reads it off the wire. Logging rules are thus created and managed independent of the code.

However, this approach can miss some important events that require getting into the code—for example, logging messages is fine, but nobody can infer exceptions from them or determine what happened to the message after sending it. Building the runtime code with a set of events to watch for and log gives much more actionable information. After all, the code is the arbiter of how the system will ultimately use the message.

An alternative approach adds a Web services security gateway between the service requester and provider that proxies the communication channel. The benefit here for audit logging services is that the gateway has access to the full message to perform logging operations. In addition, Web services gateways are used to perform application tasks such as message transformation as well as security services such as authentication and authorization. Because the gateway has access to the full message (header and body) and controls policy-based events such as authentication and authorization de-

An interesting challenge that's likewise sure to emerge is how to handle logging in distributed environments such as those of Web services.

Table 1. Audit log checklist for a Web services applications.

CATEGORY	WHAT EVENTS TO LOG
Authentication, authorization, and access	Authentication/authorization decisions, system access, data access
Changes	System/application changes (especially privilege changes), data changes (including creation and destruction)
Threats	Invalid input (schema validation failures)
Exhausted resources, capacity	Limits reached (message throughput, replays), mixed availability issues
Startups and shutdowns	Startups and shutdowns of systems and services
Faults and errors, backup success/failure	Errors, faults, and other application issues

cisions, the visibility and control of events makes it a good locale for writing audit log events.

What Should We Log?

Several types of logging events can occur in a Web services infrastructure. First, let's consider the most basic type of event: authentication. When a Web services client authenticates against a Web service, this event should certainly be logged. A typical scheme authentication event would include authentication credentials (username and password) or information about other mechanisms used for authentication (such as tokens). Due to possible use of single-sign-on, authentication itself might be distributed and happen on a different server, further highlighting the need for Web services logging to be distributed. As we mentioned earlier, Web services typically use two security models—interface and implementation security and message security—and authentication can happen in either layer independently or in different parts of the software architecture. The log event model must account for both types of event and the fact that they can occur asynchronously.

Other logging events include authorization and access. Both successful and failed attempts to get access to data or services must be logged. Although some people focus on logging just the failed attempts, successful ones are equally important: if a Web service client accesses data that it isn't supposed to, logs can provide one of the few ways to know that this has happened. Earlier, we discussed the claims-based access control model for Web services, which doesn't have a necessarily hard boundary between authentication events and authorization events. The security architect must define these event types and results for the event log model.

Other logging events include various failures, errors, and exceptions. It's difficult to create a specific list of errors and failures that must be logged, but many of the simple ones can point to potentially significant security failures—for example, a corrupted call to a Web service might indicate a software bug or an attempt to send corrupted requests for exploitation. Specifically, WS errors detected while validating the request input are critical to log because many attacks include specially formatted incorrect input (malformed input) to exploit the service's back end. Resource exhaustion issues are also important to log because they might indicate a denial-of-service attack or the consequences of a "benign" software bug. Even log evidence of an application server crash might be critical evidence for an incident investigation.

Web services are applications that are as much configured as they are coded. As such, numerous management and policy systems come into play, so key management operations, enterprise service bus (ESB) federation, orchestration, and other intermediaries must be logged as well. Table 1 gives a checklist to help guide what might be valuable to log for a Web services application.

The biggest challenge in logging is where to generate, store, and manage the log. A Windows Web service requester might write to the Windows event log, but the message goes to a Java Web service provider that, by default, writes its own log to Websphere. Even in this simple case we can see that a complete end-to-end log of something as clear cut as a single request-response interaction is a challenge.

Web services are designed to offer a loosely coupled integration. The service requester and service are independent, often running in separate technical and policy domains. In fact, in almost all cases, there's no notion of a classic two-phase commit transaction, even for high-value transactions—instead, Web services are "eventually consistent," meaning that the client and server both independently conduct computations and write events. Later, through replication, updates, and other means, these events are reconciled for greater consistency. What this approach lacks in perfection, it makes up for in its proven, massive scalability.

The net result of the Web services logging problem is both a challenge and an opportunity. The challenge is that there's no cohesion or consistency in how

loosely coupled services log these events: decisions are localized, so what to log, how to log it, and how to interpret the results are left up to each service, and various endpoints will likely do it differently. On the flip side, when the audit log browser executes the algorithms to generate the "eventually consistent events," where do they go to read what happened? They go to the logs! By being a data source of last resort, logs have achieved something close to first-class citizenship in Web services architectures.

One thing you can't say about most Web services and XML architectural concerns is that they lack standards. From a security standpoint, WS-Security, WS-Trust, and WS-SecurityPolicy provide useful, standardized tools for building a robust security architecture that supports the multiple token types, trust domains, and expressive policies that integrated systems require. However, the standards bodies have left us high and dry on logging; various attempts have launched over the years, but so far the industry hasn't adopted anything widespread. We're on our own when it comes to deciding what to log, meaning that not only do we have the challenge of where to log, but we also lack consistency in log and event types.

As Web applications and services evolve and become even more important for organizations of all sizes, taking control of Web services logging for increased accountability, security, resiliency, and regulatory standard satisfaction likewise becomes crucial. □

References

1. G. Peterson and H. Lipson, "Security Concepts, Challenges, and Design Considerations for Web Services Integration," Carnegie Mellon Univ. and Cigital, 2006; https://buildsecurityin.us-cert.gov/daisy/bsi/articles/best-practices/assembly/639-BSI.html.
2. W. Vogels, "Eventually Consistent," *ACM Queue*, 4 Dec. 2008; http://queue.acm.org/detail.cfm?id=1466448.

Anton Chuvakin *is involved with PCI DSS compliance at Qualys. His technical interests include vulnerability management, log management, and security compliance issues. Chuvakin has a PhD in physics from SUNY Stony Brook. Contact him at www.chuvakin.org.*

Gunnar Peterson *is managing principal of Arctec Group. His research interests include security in distributed systems, and software security. He is a visiting scientist at Carnegie Mellon University's Software Engineering Institute. He maintains an information security blog at http://1raindrop.typepad.com.*

ADVERTISER INFORMATION • MAY/JUNE 2009

Advertiser — **Page**
Black Hat — Cover 4

Advertising Personnel
Marion Delaney
IEEE Media,
Advertising Dir.
Phone: +1 415 863 4717
Email: md.ieeemedia@ieee.org

Marian Anderson
Sr. Advertising
Coordinator
Phone: +1 714 821 8380
Fax: +1 714 821 4010
Email: manderson@computer.org

Sandy Brown
Sr. Business
Development Mgr.
Phone: +1 714 821 8380
Fax: +1 714 821 4010
Email: sb.ieeemedia@ieee.org

Advertising Sales Representatives

Recruitment:

Mid Atlantic
Lisa Rinaldo
Phone: +1 732 772 0160
Fax: +1 732 772 0164
Email: lr.ieeemedia@ieee.org

New England
John Restchack
Phone: +1 212 419 7578
Fax: +1 212 419 7589
Email: j.restchack@ieee.org

Southeast
Thomas M. Flynn
Phone: +1 770 645 2944
Fax: +1 770 993 4423
Email: flynntom@mindspring.com

Midwest/Southwest
Darcy Giovingo
Phone: +1 847 498 4520
Fax: +1 847 498 5911
Email: dg.ieeemedia@ieee.org

Northwest/Southern CA
Tim Matteson
Phone: +1 310 836 4064
Fax: +1 310 836 4067
Email: tm.ieeemedia@ieee.org

Japan
Tim Matteson
Phone: +1 310 836 4064
Fax: +1 310 836 4067
Email: tm.ieeemedia@ieee.org

Europe
Hilary Turnbull
Phone: +44 1875 825700
Fax: +44 1875 825701
Email: impress@impressmedia.com

Product:

US East
Joseph M. Donnelly
Phone: +1 732 526 7119
Email: jmd.ieeemedia@ieee.org

US Central
Darcy Giovingo
Phone: +1 847 498 4520
Fax: +1 847 498 5911
Email: dg.ieeemedia@ieee.org

US West
Lynne Stickrod
Phone: +1 415 931 9782
Fax: +1 415 931 9782
Email: ls.ieeemedia@ieee.org

Europe
Sven Anacker
Phone: +49 202 27169 11
Fax: +49 202 27169 20
Email: sanacker@intermediapartners.de