The Security of Web Services as Software

Karen Mercedes Goertzel
Booz Allen Hamilton

To help creators of Web services and Service-Oriented Architectures (SOAs) understand and address the security challenges that confront them, the National Institute of Standards and Technology (NIST) is getting ready to publish a new Special Publication (SP) 800-95, Guide to Secure Web Services. This SP describes Web service security standards and explains how to develop Web services and SOA portals using technologies based on those standards. However, neither SP 800-95 nor the standards it describes address a critical challenge: the security of Web services as software. Without considering software security, developers cannot create Web services that are truly trustworthy. This article describes both the content of SP 800-95 and highlights its critical omissions in terms of measures needed to produce Web service software that is in and of itself secure.

Web services in SOAs allow applications to interact and data to be interchanged without direct human intervention. The use by Web services of eXtensible Markup Language (XML), SOAP (formerly Simple Object Access Protocol), and related open standards enables these interchanges to be achieved over connections that are established dynamically on an *ad-hoc* basis.

Web service-based SOAs are proliferating exponentially, both in number and extent, in Department of Defense (DoD) and other government agencies, the private sector, and academia. Web services are relied on to create, manipulate, and protect information, including the most sensitive private information. They are used to perform high-consequence and mission-critical information-handling functions.

The individual, software-intensive Web services of which these SOAs are composed are also growing larger and more complex, to the extent that their own developers can no longer fully recognize – let alone comprehend – all of their possible behavioral states, weaknesses, and vulnerabilities.

At the same time, software-level threats to Web services and portals to SOAs are increasing in intensity, sophistication, and variety. Zero-day vulnerabilities in commercial Web service software products have not only been proven to exist, they are on the rise. (A zero-day vulnerability is one against which an exploit is launched by an attacker before a security patch can be issued by the product's developer.)

The security challenges presented by the Web service processing model, in which SOAP messages encapsulating sensitive XML documents are forwarded along complex chains of consumer, provider, and intermediary Web services, are formidable. This is because many of the features that make Web services and SOAs attractive – greater accessibility of data, dynamic establishment of interservice relationships and communication paths, a high degree of service autonomy, and a minimal amount of direct human involvement – are all at odds with traditional security models and controls.

The nature of Web services and SOAs makes them subject to unique attacks, as well as variations (often involving intensification) of familiar attacks that already target Web servers and applications. Achieving secure Web service processing entails the extension and augmentation of existing Web and Internet security mechanisms. This is increasingly achieved through the implementation of authentication, authorization, trust, confidentiality, integrity, and availability of functions based on relatively new and still-emerging Web services security standards.

NIST SP 800-95, Guide to Secure Web Services

To help the architects, developers, and engineers involved in the creation of Web services and SOAs understand and address the security challenges that confront them, the NIST is publishing a new SP 800-95, *Guide to Secure Web Services*. The current draft of SP 800-95 can be downloaded from the SP page of the NIST Computer Security Resource Center Web site: <http://csrc.nist.gov/publications/nistpubs/>. The final version will be published at this address later this year.

The objective of this NIST SP is to describe and make sense of the broad range of Web service (WS) security standards that have been produced by the Organization for the Advancement of Structured Information Standards, World Wide Web Consortium, Web Services Interoperability Organization (WS-I), the Liberty Alliance, and other standards bodies. Because of this objective, the SP tends to focus on security issues that are already being addressed by these various standards bodies and omits discussion of those security challenges that have not yet been acknowledged by the standards community, or that the community has deemed *unimportant*, *out of scope*, or *too hard* to address via Web services security standards.

While SP 800-95 does describe how to implement security functions and protections for Web services and SOA portals, it does so almost exclusively from the point of view of what can be achieved using technologies based on current and emerging Web services security standards.

Unsurprisingly, after providing background information on key Web services and SOA concepts, capabilities, and components, including discovery, messaging, portals, coordination (choreography and orchestration), and the roles, modes, and properties of Web services, the SP's discussion is limited to what is already widely accepted as constituting *Web services security*: secure interservice messaging, protection of XML-based content, secure negotiation of contracts between service providers, and establishment of trust relationships between services.

For each of the security capabilities and protections it introduces, the SP describes the associated Web services security standards. These standards are organized into a stack (comparable to the Open Systems Interconnect [OSI] and Transport Control Protocol/

Internet Protocol network protocol stacks). Within an SOA, the standards are also frequently implemented within core security services, i.e., security services that other services depend on to perform essential functions on their behalf. The SP also describes the security threats to Web services that protections and functions based on these standards are intended to address. These functions and protections fall into eight general categories:

1. **Authentication and identity management.** The SP addresses service-to-service authentication and service, and identity management and authentication of human users who access SOAs via Web portals. Specific standards discussed include WS-Security for interservice authentication (including its security shortcomings), and Security Assertion Markup Language (SAML) as the basis for single sign-on by human users.
2. **Interservice trust.** The SP addresses trust establishment (based on authenticated identity) between services, and federation of trust across SOA boundaries.
3. **Secure service discovery.** The SP focuses on security for Universal Description, Discovery and Integration (UDDI) and Web Service Description Language (WSDL) interfaces, secure access to the SOA registry, secure interaction of Web portals with the SOA discovery services, and application programmatic interfaces (APIs) for secure service inquiry and publishing.
4. **Distributed authorization and access management.** The SP discusses the authorization of privileges to Web services and the enforcement of least privilege in SOA authorization models. Specific standards discussed include SAML as the basis for asserting privileges, eXtensible Access Control Markup Language (XACML) as the basis for service-level access control, and the use of XML schema and security metadata for data/content-level access control.
5. **Confidentiality and integrity of service-to-service interchanges.** The SP discusses use of HyperText Transport Protocol Secure (HTTPS) for transport layer security in SOAs, WS-Security for SOAP message-level security, XML encryption and signature for content protection, and the role of XML gateways in providing additional message-level and content-level integrity protection.
6. **Accountability.** The SP discusses methods for achieving accountability end-to-end throughout a Web service chain, including audit in SOA environments and use of XML signatures as the basis for non-repudiation of Web service transactions.
7. **Availability and quality of service (QoS).** The SP discusses availability and QoS techniques such as fail-over, handling of service deadlocks and service recursion, and it addresses the security implications of competing reliable messaging standards.
8. **Security policy specification.** The SP discusses WS-Policy and its role in supporting specification and enforcement of security policy within an SOA.

Recognizing that standards are only the basis for implementations, the SP touches on tools and technologies for implementing security standards-based Web services, such as Web service-oriented developer toolkits, XML parsers, procedural languages commonly used in Web service development, XML, and tools and techniques for testing the security of Web services.

> *"What Web service software does share with other software is its exposure to threats throughout its lifetime, not just after it has been deployed, but while it is still under development."*

In this context, the SP also addresses issues associated with secure Web service enabling of legacy applications (e.g., public-key enabling consistent with Web services standards and implementing standards-based security functions and protections for legacy applications and databases exposed as Web services).

Security of Web Services as *Software*

Though SP 800-95 does cover implementation of Web services that are likely to be called secure because they are based on accepted Web services security standards, the SP does not discuss what is probably the most important security issue in the implementation of Web services – an issue that is not addressed by any Web services security standard: The security of Web services as software.

It can be argued that because Web services are subject to the same security issues as all software, and particularly network-based application software, no discussion of the topic was needed in NIST's Web services security guidance. Indeed, NIST excluded such a discussion as out of scope exactly because the need for software security is not unique to Web services.

This is an interesting position for NIST to take, given that one can argue equally that requirements for confidentiality, integrity, availability, authentication, trust, identity management, etc., are not unique to Web services either. Moreover, in several cases the standards that are being used to address these requirements were never intended to be exclusive to Web services. For example, XML digital signature and encryption are intended for use with all XML documents, not just those exchanged between Web services. SAML and XACML, while certainly used to implement authentication and access control in Web service implementations, are intended to be widely applicable to all types of applications.

Threats to Web Service Software

What Web service software does share with other software is its exposure to threats throughout its lifetime, not just after it has been deployed, but also while it is under development. Threats to software in development may be intentional, coming from malicious developers who subvert or sabotage the software they or their colleagues build, or they may be unintentional, introduced by developers who, due to ignorance, carelessness, or pressure to get the software out on time fail to implement checks and controls that would eliminate or minimize exposure of the software's numerous weaknesses and vulnerabilities.

Compounding this problem is the almost universal use of commercial and open source software components in the building of Web services. The pedigree of such components is often a mystery. Neither the processes used to develop those components, nor the security properties, weaknesses, and vulnerabilities of the components themselves are ever investigated by the developers who

incorporate those components into their Web services. Even pedigree of open source components is taken entirely on faith. Just because open source code is available for review does not mean that anyone actually does code reviews of open source code.

Specific threats to Web services under development fall into two general categories: 1) malicious code, and 2) exploitation of weaknesses and vulnerabilities.

1. **Malicious code.** Logic bombs, time bombs, Trojan horses, worms, and other undocumented malicious functions are intentionally inserted into the source code or appended to the binary executable by the developer. Failure to review source code and carefully observe the behavior of executing binary code means that such embedded malicious code will be allowed to remain in software when it is deployed. Furthermore, malicious code may be added by an attacker who intercepts and tampers with electronically distributed executables. After the software is deployed, it is subject to the delivery of new malicious code or the execution of embedded malicious code. The risk of malicious code insertions and executions is increased when the software and its environment are not configured insecurely, and anti-malicious code countermeasures are not deployed or used effectively.
2. **Exploitable weaknesses and vulnerabilities.** Flaws, defects, errors, and faults are often included, usually unintentionally but sometimes intentionally, in the artifacts – specification, architecture, design, or implementation – of the Web service. In some cases, these can be intentionally leveraged by attackers (or by malicious software processes acting on an attacker's behalf) to compromise the security of the Web service itself or of the data it handles.

Weaknesses originate as early as the requirements phase. Security-related requirements may be overlooked or misstated, or spurious requirements may be included. They may arise as the result of poor architecture or design choices, such as failure to enforce least privilege or to design in redundancy of critical processes.

Vulnerabilities may enter software during its implementation, due to use of non-secure implementation practices. Examples of such practices include: accepting user input without first validating it; using vulnerable technologies (e.g., SOAP over unencrypted, unauthenticated HTTP connections); use of non-secure programming languages (e.g., non-type-safe languages used without input validation) and library functions (e.g., buffer overflow-prone C functions such as printf); use of non-secure development tools (e.g., compilers that do not perform bounds checking); reuse of vulnerable components (e.g., commercial software that has known vulnerabilities); use of development tools (e.g., compilers that do not perform bounds checking); or reuse of vulnerable components (e.g., commercial software that has known vulnerabilities).

Weaknesses and vulnerabilities may be allowed to remain in software due to the failure to perform adequate security reviews, assessments, and tests of the artifacts of the development process (from specifications through the software itself); or the intentional tampering with the results of such reviews/assessments/tests.

They may also be allowed to remain in the form of back doors and trapdoors that are not removed prior to software distribution. They may arise or fail to be mitigated due to specification of non-secure configuration parameters for the software and its environment (or the use of non-secure installation procedures, scripts, and tools).

Once the Web service is operational, it is subjected to misuse by its intended users, and abuse by attackers. Understanding the attack patterns to which Web services are likely to be subject can be extremely helpful to the developer in specifying security requirements, architectural characteristics, and design properties that can reduce a service's exposure and vulnerability to likely attack patterns. Moreover, such understanding provides a basis for defining the code assessment criteria and security test plans for developmental and non-developmental Web service software components (i.e., attack surface definitions highlight specific targeted security flaws to look for during code reviews; misuse and abuse cases can be elaborated into white-box and black-box test scenarios).

Exploits Against Web Services

The exploits, or attacks, that target existing Web services fall into two main categories: direct and indirect. Direct attacks exploit known or suspected vulnerabilities and weaknesses in the Web service itself, while indirect attacks may target the service's interface with the environment and middleware components on which it relies, or its interface with the external services and applications with which it interacts.

Attacks against Web services have one of three general objectives:

1. **Disclosure.** This may be achieved through reconnaissance attacks that discover or reveal Web service vulnerabilities that can be exploited by other attack patterns. It may also be accomplished through attacks that bypass or cause denial of service in the Web service so as to directly access and disclose the sensitive/private data handled by that service.
2. **Subversion.** This includes subversion of the service's functionality (i.e., by direct tampering, malicious code insertion/delivery, command injection, tampering with the state of the service's execution environment, and intentional triggering of errors or faults at the service boundary with its environment), of its data, or of its security assumptions about other services (i.e., by an attacker or malicious process masquerading as an entity or hijacking an entity that is trusted by the service, and thereby escalating its own privileges to match those of the trusted entity). It also includes attacks that bypass or cause denial of service in the service in order to directly access or tamper with the data the service handles.
3. **Sabotage.** This may be achieved through denial-of-service attacks on the service itself, or on the external entities on which it depends for its dependable, secure operation (e.g., execution environment and network components, core security services, and defense-in-depth protections). An objective of sabotage is often to bring down or bypass the targeted software in order to directly access the data in control.

Particular Security Challenges for Web Services in SOAs

Some security challenges are unique to Web service software, and others are greatly exacerbated when they arise in Web service software. The chains of dependencies between autonomous, dynamically invoked Web services within SOAs are often much more complex than when autonomous software components are used in more traditionally

distributed processing models. In most distributed systems, the list of components that will be invoked, and the order in which they will be invoked, in the course of completing a particular transaction or task is predefined, as to a great extent are the outputs of the invoked components. In an SOA in which services are dynamically coordinated (through choreography or orchestration), it is frequently impossible to predict in advance which services will be invoked by other services, and in what order those invocations will take place. In dynamic coordinations that cross the boundaries from one trust domain to invoke services in another trust domain, it is especially hard to establish valid security assumptions in advance about the behaviors, policies, and permissions expected by the services in the remote domain. If a fault in a Web service causes that service to violate expected behavior or policy, the results of such a violation have the potential to propagate throughout the entire chain of services. Because that chain is unpredictable (being dynamically established rather than pre-defined), the propagation and impact of the violation will also be unpredictable. The result of a fault in one Web service, then, may compromise the security and dependability of other Web services much further along the chain, which may make forensic analysis to identify the true source of the compromise and to trace all the possible branches of its progress extremely difficult (if not impossible).

Moreover, in SOA implementations, each service is inherently dependent on other autonomous services. The increasingly widespread use of what are termed *core security services* model means that many innately non-secure Web services depend on other services for critical security protections and capabilities. Their role as security service providers means that these core services are not only the most critical services in the SOA, but represent the highest-value targets to attackers.

Even when provided via core security services, the SOA's security functions and protections are often actually implemented using the security functions and protections provided by the underlying application framework, e.g., Java Enterprise Edition or .NET. This increases the risk that consumer Web services not based on the same framework technologies may not interoperate seamlessly with the core security services. The centralization of the SOA's security functions into a set of core services increases the imperative to ensure that such services, which will be trusted to guarantee the entire SOA's security posture, will be able to resist or tolerate attacks and to continue operating reliably under hostile conditions. If such software contains weaknesses and vulnerabilities that can be exploited by attackers, this model collapses due to the misplacement of trust in components that are too vulnerable to perform their designated tasks.

A number of other factors unique to Web services and SOAs make their software components more vulnerable to software-level exploits than other types of application software. First and foremost among these are the following:

1. The woefully inaccurate assumption that Web service interfaces will be used only as intended by other Web services, and not by human beings or malicious processes. Because Web services generally have no direct human interfaces, their operation receives little if any human scrutiny or intervention ... *except* by attackers.
2. The unavoidable fact that by exposing Web services applications and databases that were never originally intended for direct public access, the weaknesses and vulnerabilities of those applications and databases are also exposed to public view. Moreover, the easy-to-use development tools used by many Web services developers obscure the services' low-level functionality from the developer; it is these low-level functions that often contain the exploitable weaknesses and vulnerabilities that are targeted by attackers and malicious code. For example, Apache Axis 2 enables a Java developer to simply load his/her Java objects into the Axis SOAP engine. At runtime, it is the SOAP engine that determines which incoming SOAP request messages should be routed to which Java objects. The SOAP engine then translates those requests into standard Java function calls and routes them appropriately. Unless he/she has expressly reviewed the source code of the Axis SOAP engine, he/she will have no idea whether its routing or translation functions contain embedded malicious logic that could result in incorrect routing of messages or incorrectly generated Java calls. In the case of a commercial tool, such as Visual Studio, the ability to review the tool's source code is not even an option.

> *"Understanding the attack patterns to which Web services are likely to be subject can be extremely helpful to the developer in specifying security requirements, architectural characteristics, and design properties that can reduce a service's exposure ..."*

Automatic discovery of Web services in particular is a feature of SOAs that makes it easier for attackers to locate and access potential targets. Publishing of repository entries about services through standard discovery interfaces (WSDL and UDDI) represents an unprecedented level of public disclosure of service processing details – details that can be used by reconnaissance attackers to craft much more effective attacks on the discovered services. Moreover, to accomplish automatic service discovery, privileges must be granted to unknown entities outside the organization that owns the services being discovered. There is a significant question as to the extent that entities outside the SOA, which interact with services discovered inside the SOA, can be governed by the policies (including security policies) enforced for that SOA.

For example, Organization #1, which operates SOA #1, may mandate that all Web service software must undergo code review and penetration testing before being deployed. Organization #2, which operates SOA #2, may have no such policy. So, if SOA #1 establishes a federated trust relationship with SOA #2, there is no way for Organization #1 to know whether the Web services in SOA #2 contain exploitable faults or malicious logic. Trust, as it is defined for Web services, (e.g., WS-Trust) refers solely to the assurance that a given service's identity has been authenticated by a trusted third

party. By this definition, trust has nothing to do with the authenticated service's *trustworthiness*. The non-malicious functioning and behavior of the service must be taken entirely on faith.

Finally, all of the security problems associated with component-based software systems are also present in service choreographies and orchestrations, and furthermore, exacerbated by the increasingly *dynamic* nature of service composability. While security assumptions about individual services may be derived from the services' WSDL descriptions, and when services are combined in ways that differ from transaction to transaction, it is virtually impossible to establish (1) whether the security assumptions about a given service are still valid and meaningful when that service is instantiated within a given choreography/orchestration, and (2) whether there are any irresolvable security conflicts between services that are dynamically composed into a choreography or orchestration.

It is true that some features of Web service technology actually help mitigate security issues found in other types of software. Most notably, the reliance of Web services on platform-independent, standards-based APIs such as WSDL and SOAP, rather than using proprietary and/or platform-specific APIs, makes it easier to replace vulnerable Web services quickly with less vulnerable substitutes. Use of standard APIs also enables diversity of service implementations – a secure design principle that, when coupled with redundancy of services, reduces risk by reducing the number of services that will be compromised by an attack pattern tailored to exploit a specific vulnerability in a particular Web service implementation (or product). The result is an improvement in availability because the alternate services are unlikely to be susceptible to the same implementation-specific attack patterns that compromised the services they back up.

Building Secure Web Service Software

What can be done to make Web service software trustworthy? In practical terms, trustworthiness will be achieved by producing a Web service that is dependable, not only under both expected operating conditions but also under unexpected and intentionally hostile operating conditions. It is this dependability under unexpected hostile conditions that constitutes software *security*.

In practical terms, intentionally hostile operating conditions are created either by the presence of attack patterns or the behaviors that result from execution of malicious code. To continue operating dependably, then, a Web service must be designed and implemented so that it is able to do the following:

- Recognize and *resist* or block most attack patterns and malicious behaviors.
- *Tolerate* and safely handle the errors and failures that result from those attacks and malicious behaviors it cannot resist or block.
- Exhibit resilience by isolating and constraining the damage and recovering quickly (to an acceptable level of capability) from successful attacks and malicious behaviors.

> *"... intentionally hostile operating conditions are created either by the presence of attack patterns or the behaviors that result from execution of malicious code."*

Furthermore, to be deemed trustworthy, Web service software must not only exhibit the properties that constitute dependability and security, but also those that constitute assurability, which is the ability to independently verify the software's other required properties. The properties that constitute software dependability are correctness and predictable execution (i.e., the software does what it is supposed to do and nothing else), and in some cases safety. (Software safety is defined in the National Aeronautics and Space Administration Software Assurance Glossary <http://sw-assurance.gsfc.nasa.gov/help/glossary.php> as the systematic identification, analysis, tracking, mitigation, and control of software hazards and hazardous functions, hazards being existing or potential conditions or functions that can contribute to or result in mishaps or accidents. Software safety does not concern itself with preventing intentionally induced incidents, even though such an incident could result in a mishap or accident.)

The properties that constitute software security are *integrity* (inability to subvert) and *availability* (inability to sabotage), and for Web services, *accountability* of the service as a non-human actor in a SOA (which includes *non-repudiation* by the service of its actions). In many cases, confidentiality of the software itself is also a desirable security property: The software's executable and/or operational behaviors may be hidden and/or obfuscated to make reconnaissance and disclosure of vulnerabilities difficult.

The properties that promote assurability include the following: *simplicity* (of design and implementation), *smallness* (of code), and *traceability* (of implementation to requirements).

In practical terms, a Web service can be said to be secure when it achieves the following:

1. The behavior of the service itself (including its behavioral state changes in response to inputs and external events) does not make the service vulnerable to attack or malicious code insertion/execution. This means the service must handle all inputs safely, validating them before use and rejecting or modifying (to make acceptable) those that threaten its secure behavior. It also means that the service must handle errors, exceptions, faults, and failures safely and securely, so that these events do not cause the software to enter an insecure state or compromise the data and resources to which it has access.
2. The service's interactions and interfaces must be secure. This includes those used among the service software's constituent components, e.g., remote procedure calls, and those used between the service and any external entities, including other Web services (e.g., SOAP over HTTPS with WS-Security), environment-level components (i.e., APIs, call-level interfaces), and human users (i.e., user interfaces).
3. The service is executable and data files in the file system must be protected from unauthorized access. This means that the configuration parameters of the service itself and of its execution environment protections must be as restrictive as possible.
4. The service's attack surface must be minimized. If this has not been achieved by reducing the number of vulnerabilities in the service itself (both at the architectural and software levels), it might be achievable through defense-in-depth measures that minimize the exposure of the vulnerabilities that were not eliminated.

Conclusion

While NIST's new SP 800-95, *Guide to Secure Web Services*, should prove helpful in increasing Web service implementers' knowledge and understanding of the security standards being adopted to secure service-to-service interactions within distributed SOA-based information systems, the SP does not discuss methods and techniques for design and implementation of secure Web service *software*. This leaves it up to the Web service developer to find that type of information elsewhere.

A good place for developers to start looking is the Department of Homeland Security's (DHS') BuildSecurityIn Web portal at <https://buildsecurityin.us-cert.gov/>. The resources here provide a broad range of recommendations on how Web service developers can add security principles and practices to their existing software processes so that the software produced by those processes will not only perform its required security functions, but will exhibit the levels of attack-resistance, attack-tolerance, and attack-resilience required to minimize its attack surface and susceptibility to malicious code penetrations and executions.

Recently, the Defense Information Systems Agency (DISA) published a Security Technical Implementation Guide entitled Application Security and Development Security. This can be downloaded at <http://iase.disa.mil/stigs/stig/asd-stig.pdf>.◆

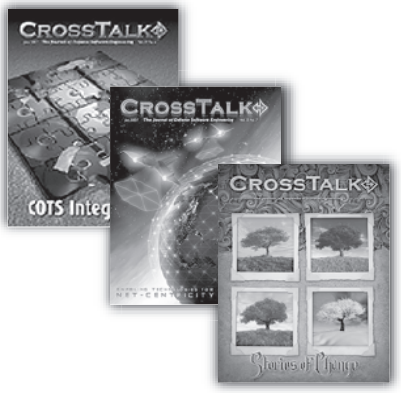
About the Author

Karen Mercedes Goertzel is a software security subject-matter expert supporting the Director of the DHS Software Assurance Program and has provided similar support to the DoD's Software Assurance Tiger Team. From 2002-2004 she was project manager of the DISA Application Security Support Task and is currently leading the team developing NIST Special Publication 800-95, *Guide to Secure Web Services*. In addition to software assurance and application security, Goertzel has extensive expertise and experience in trusted systems and cross-domain information sharing solutions and architectures, information assurance (IA) and cybersecurity architecture, strategy and planning, risk management, and mission assurance. She has written and spoken extensively on software security and IA topics, both in the U.S. and abroad.

Booz Allen Hamilton
8283 Greensboro DR H5061
McLean, VA 22102
Phone: (703) 902-6981
Fax: (703) 902-3537
E-mail: goertzel_karen@bah.com

CALL FOR ARTICLES

If your experience or research has produced information that could be useful to others, CROSSTALK can get the word out. We are specifically looking for articles on software-related topics to supplement upcoming theme issues. Below is the submittal schedule for three areas of emphasis we are looking for:



Project Tracking
April 2008
Submission Deadline: November 16, 2007

Software Safety
May 2008
Submission Deadline: December 6, 2007

Information Assurance
June 2008
Submission Deadline: January 18, 2008

Please follow the Author Guidelines for CROSSTALK, available on the Internet at <www.stsc.hill.af.mil/crosstalk>. We accept article submissions on all software-related topics at any time, along with Letters to the Editor and BACKTALK. Also, we now provide a link to each monthly theme, giving greater detail on the types of articles we're looking for <www.stsc.hill.af.mil/crosstalk/theme.html>.

COMING EVENTS

October 2-4
STPCON 2007 Software Test and Performance Conference
Boston, MA
www.stpcon.com

October 16-17
ICSQ '07International Conference on Software Quality
Denver-Lakewood, CO
www.ndia.org

October 22-25
10th Annual Systems Engineering Conference
San Diego, CA
www.ndia.org

October 22-26
STARWEST 2007
Software Testing Analysis and Review
Anaheim, CA
www.sqe.com/StarWest/

October 29-30
VERIFY 2007
Crystal City, VA
http://verifyconference.com

November 4-7
AYE 2007
Amplifying Your Effectiveness
Phoenix, AZ
www.ayeconference.com

May 2008
Systems and Software Technology Conference
www.sstc-online.org

COMING EVENTS: Please submit coming events that are of interest to our readers at least 90 days before registration. E-mail announcements to: nicole.kentta@hill.af.mil.