

Alone in the Dark: The Perils of Securing Web Services

John Carmichael

*Security Analyst
Security Innovation*



Security Innovation, Inc
187 Ballardvale Street, Suite A170
Wilmington, MA 01887

1.978.694.1008

www.securityinnovation.com

Boston

Amsterdam

Seattle

Service oriented architecture (SOA) has progressed from an industry buzz word to a preferred design principle for business systems – igniting a system design shift that gave birth to the powerful tool known as Web services. Web services are modular chunks of functionality that organizations publish and allow trading partners to access. Many of today's popular Web applications use Web services as the behind-the-scenes engine for their more complex functionality. This raises the question: How do we secure these new interfaces we are developing?

The Big Gun Threats

In order to secure something you need to first understand the threats to which it may be vulnerable. Web services have an interesting threat profile. They are standard pieces of functionality, typically written in .NET or Java, and often connect to file systems and databases like the programs we are accustomed to writing. As a result, Web services are not exempt from the major threats that we concern ourselves with when securing traditional software. Attack vectors like the buffer overflow, SQL injection and other parameter tampering threats, also apply to Web services. However, Web services introduces a few more, including:

WSDL Scanning. A WSDL (Web Services Definition Language) is used to describe the Web service to connecting parties. Our trading partners use these documents to discover what pieces of functionality are available to them and how to format their requests to the Web service. Care needs to be taken when creating and publishing these documents. Often the documents are automatically generated from the code and functionality not meant to be exposed to outside entities is included in our WSDL. This may allow an attacker unintended access functionality.

XPath Injection. XPath is a language for querying information from XML documents. Similar to SQL Injection, if user input is not properly sanitized, it is possible for a malicious user to influence the XPath query being run by the software to garner more information than he/she would normally have access to.

Recursive Payload. The communication sent back and forth via Web services is all XML base, giving the attacker a new avenue of attack. Knowing that the Web service will need to parse the XML message in order to process the request, an attacker can send a request which contains a large amount of nested opening tags, but never supply a closing tag. The Web service, when trying to parse this file, will often consume too many system resources or even crash as it needs to track open tags until the matching close tag occurs. This can cause a denial of service to the Web service.

Opening pieces of functionality to third parties is wrought with threats, both old and new. For this reason it is paramount that developers understand these threats and how to protect their applications from potential attack. The biggest roadblock to securing Web services is understanding that it is difficult to do so.

The CIA of Web Services Security – Confidentiality, Integrity, Availability

The three tenets of security are confidentiality, integrity and availability. In the world of Web services, availability is the most straightforward to achieve. Typical attacks against Web services availability would be based on bad data, which is determined to choke the application and cause it to crash. Developers need to define strict rules for their input to act as guidelines for validation. Any and all data is then validated against these rules prior to use by the system. This will help protect against availability attacks. Although protecting the availability of Web services is no simple task, it is much easier than protecting confidentiality and integrity.

With confidentiality, we want to ensure that only the intended audience is able to access information. Integrity means that we know where the data came from and that it has not been altered in transit. For both of these we need to have strong authentication that allows the system to validate a true identify and authorization which grants access permission to only authorized users. When we attempt to implement these measures in Web services we find ourselves falling down a rat hole of acronyms and cobbled pieces which only address part of the issue.

The WS-Security Standard for Message Based Security

Often the first place security professionals tend to look for help is the WS-Security (Web Services Security) standard. WS-Security is a proposed standard for dealing specifically with confidentiality and integrity for Web services. I've seen many implementations of Web services which attempt to sprinkle magic SSL/TLS security dust on the problem to make it go away. While using a protocol like HTTPS to transmit the messages between the requestor and the Web service, this only provides point-to-point security and does not address security for the message after it reaches the other point. We need so called end-to-end security and WS-Security attempts to provide us with that.

WS-Security allows us to attach timestamps to messages to ensure freshness and prevent replay attacks. There is a mechanism included for encrypting messages which provides the needed confidentiality. There is also a mechanism for digitally signing messages, which authenticates the sender and ensures the message has not been tampered with—meeting our integrity requirement. It also allows us to attach security tokens to a message such as username/password or X.509 certificates which can be used for authentication.

While a step in the proverbial right direction, WS-Security does have drawbacks such as performance issues and key management and distribution concerns. The most glaring however, is that it does not provide any authorization to know if the requestor has access to the information and functionality they are requesting. For this we can link in SAML (Security Assertion Markup Language), turn to XACML (eXtensible Access Control Markup Language); or use both.

Other Standards Boost Access Control

SAML and XACML attempt to provide a means to create access control policies that can be enforced by the system. This allows restricted access to certain data and functionality based on a requestor's identity. Both SAML and XACML can create policies which describe proper access controls for data and operations. The problem is that these access control policies are not easy to create, understand or manage. It is also difficult to determine which language to use.

As inferred from the challenges mentioned above, securing a Web service is a daunting task. Unfortunately, there is little help for developers in determining how best to integrate these components into their programs. Developers are left to flail about, hoping to stumble upon secure implementations. The typical response I've seen is to either delegate security to the network appliances or to ignore it all together. Neither of these options presents a desirable situation. As a security community we have to make it easier for developers to create secure code. It is our responsibility to shed light on the issue and not to leave them alone in the dark.