

Enforcing Distributed Data Security via Web Services

Alfred C. Weaver
Lucian Carr III Professor of Engineering and Applied Science
Department of Computer Science
University of Virginia
Charlottesville, VA 22901 U.S.A.

Abstract

As the manufacturing and process industries become more intelligent and more distributed, the need for reliable, secure, and verifiable data exchange becomes more acute. We have developed an approach to distributed data security based upon web services. Each web service specifies its authentication and authorization policies using standard WS-Policy combined with our novel concepts of trust levels and trust level mappings across domains. An authentication web service verifies human identity via biometric and other digital techniques; software applications are vetted by digital signatures. An authorization web service enforces a dynamic, context-aware access policy. Federation is used to manage trust relationships across separate but cooperating trust domains.

1. Motivation

A recurring theme for intelligent, distributed manufacturing is the exchange of code and data in a uniform and verifiable way. When either a human or a software application requests process data for purposes of monitoring or control, and likewise whenever any software is installed, there is the risk of a security breach—and the more distributed the system, the more difficult it is to guarantee the integrity of the overall system. The human could be an imposter; the software upgrade could contain a virus. The key issues are:

- *Authentication* – who is making the request?
- *Authentication trust level* – what is the reliability of the user's identification?
- *Authorization* – is this user permitted to read, write, change, or delete this data?
- *Federation* – how can identity, once legitimately established in one system, be safely exported to another cooperating system?

2. Why a Web Services Approach?

First, we think that Internet-accessible information is the clear wave of the future, *provided* that such access is reliable, dependable, and authentic. The Internet provides a powerful, standardized, world-wide, ubiquitous communications mechanism whose benefits are impossible to ignore.

Second, we think that wireless and mobile access technologies will become more ubiquitous in factory communications. Cell phones, pagers, PDAs, laptops, and tablet PCs are already in common commercial use, but their use within the factory enterprise has been limited by concerns over user authentication and unauthorized data disclosure. Data requests from wireless devices must be subjected to the same or higher standards for authentication and authorization as those that originate from wired devices.

Third, the web services community has already made great strides in defining the framework, standards, and languages needed for web service interactions. As promoted by the World Wide Web Consortium, web services are now seen as the preferential way to link applications both within and without an organization in a loosely-coupled, language-neutral, platform-independent way. A web services approach enables designing, publishing, promoting, registering, and initiating processes dynamically in a distributed computing environment.

3. Security Infrastructure

As illustrated in figure 1, a wireless PDA is used to access the factory data portal and display real-time process parameters; data values are retrieved from the factory data web service. How do we know that the requestor is who he purports to be? Is this individual allowed to read or modify the requested data?

A WS-Policy document defines what authentication tokens are acceptable as proof of identity for login. Upon initial access (arrow 1), a user is redirected to the authentication web service (2) to establish identity and generate an authentication token (3); the token is stored on the access device as a cookie (4), signed with the digital signature of the Secure Token Service (STS). Authentication tokens are presented automatically upon subsequent logins. Each token is valid for a limited time; token expiration forces a revalidation upon subsequent login. Portal applications, as opposed to humans, attempting to access data use digital signatures to authenticate their origin.

After successful login, all portal data requests are sent to the factory data web service (5) along with the user's authentication token. The portal's WS-Policy document defines the allowable or required authentication tokens

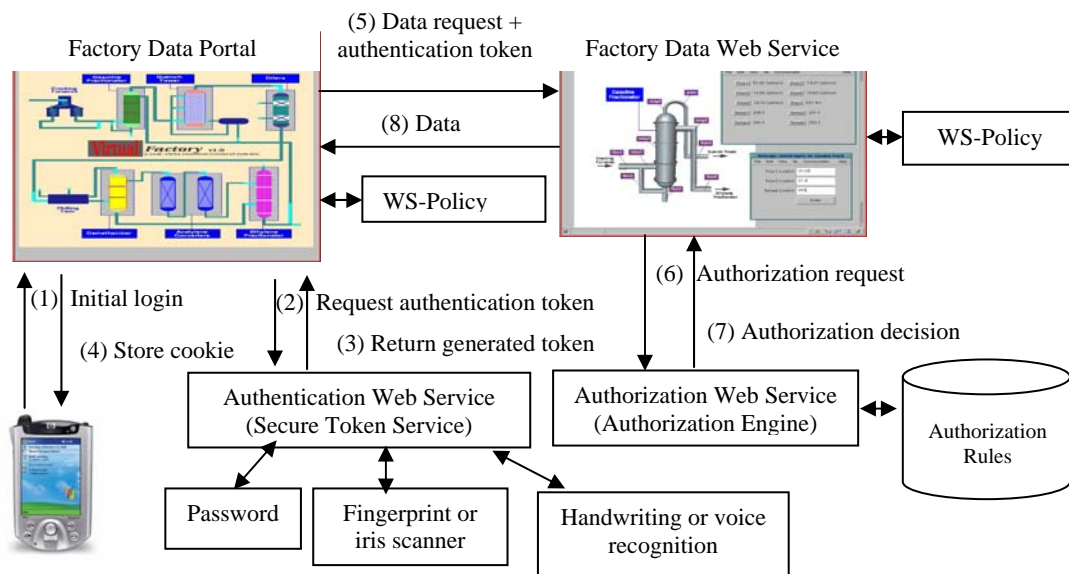


Figure 1. Authentication Process for Remote Data

to be included with all data requests, while the factory data web service's WS-Policy document defines both the authentication and authorization requirements for data access. All WS-Policy documents are XML-based.

If the data service's WS-Policy is simple, it can be enforced automatically using Microsoft's Web Service Enhancements 2.0; if the policies are complex, as ours are, then WSE 2.0 can call our own authorization web service that supports custom policy assertions. In our case, the authorization engine consults its XML-based authorization rule database to determine what permissions are to be given to a particular user when attempting to touch a protected object. Whether enforced by WSE 2.0 directly or by our authorization engine, the net result is that each data access request can be vetted for compliance with arbitrary authentication and authorization policies (all expressed in XML in WS-Policy documents). The authorization engine returns an "access permitted" or "access denied" decision based upon the user's identity, the user's role, the object being accessed, and the local context surrounding the access (e.g., did this request originate from a wireless device?).

4. Authentication

4.1 Authentication Techniques

For software applications, authentication is provided via digital signatures. For individuals, the portal can support both biometric and other digital techniques. For biometrics, we currently support the U.are.U fingerprint scanner, Panasonic Authenticam iris scanner, and CIC's iSign signature recognition software running on a tablet PC; we are investigating the new wireless Privaris

fingerprint scanner with built-in Bluetooth communications and Voice Security System's voice recognition software. For non-biometrics, we have implemented passwords, Aladdin System's e-tokens (flash memory on a USB port that supports encrypted storage of an authentication token), and the RSA SecurID hardware/software system that generates a new random number every minute on a wearable device. The RSA system requires a password ("what you know") plus the correct random number at the moment of login ("what you have"). We are working on adding a Precision Dynamics RFID capability that can be used in wrist bands or ID cards.

4.2 Authentication Trust Levels

We have extended WS-Policy and WS-SecurityPolicy to support our novel concepts of trust levels, federation, and trust mapping within the existing web services architecture. We have defined a generic format for all authentication tokens that includes the concept of trust level – that is, a numeric representation of the underlying reliability of the authentication technology. This allows the factory data web service to support an authentication policy such as "authentication requires a trust level of fingerprint or higher."

The current WS-Policy implementation in WSE 2.0 supports simple authentication policies such as "require an X.509 certificate" or "require a Kerberos ticket." By using our authorization engine, we can enforce custom policies such as "require authentication from a wired device within the enterprise to be at the trust level of a password or higher, but access from any wireless device requires authentication at the level of a fingerprint or

higher.” A major advantage of our approach is that if identity has been previously established with a higher reliability technique, that higher-trust authentication token can be used as a substitute for a required lower-reliability one without forcing the user to undergo a secondary authentication procedure.

4.3 Setting Authentication Trust Levels

The utility of this more general scheme that accepts tokens based upon trust level (while still permitting static enumeration of specific acceptable technologies, as is currently done) depends upon having an agreement about the trust level $T()$ to be associated with any particular authentication technique. In the abstract, trust levels are ordered based upon the number of degrees of freedom inherent to the underlying identification technology. For example, there is general agreement that $T(\text{retina}) > T(\text{iris}) > T(\text{fingerprint}) > T(\text{password})$.

In practice, the trust level of any specific product must be determined by experimentation to quantify its false acceptance and false rejection rates (the false acceptance rate is the percentage of authentication attempts by a person other than the enrolled individual which are nevertheless successful; the false rejection rate is the percentage of authentication attempts by the enrolled individual which are nevertheless rejected). Thus the trust levels of two separate products of the same technology, say two fingerprint scanners, may be different due to their internal implementation details. As long as the trust level of a new device can be determined satisfactorily (primarily by testing, but also taking into account industry perception and manufacturer’s reputation, or other factors deemed important within the local trust domain), it can be expressed numerically. The absolute numbers assigned to the trust levels are not important; only their relative order is used.

Within an enterprise such as a factory (the local trust domain), the trust levels of the various technologies are set by systems administrators based upon their own criteria. For example, one might order the trust levels of the technologies in the abstract along a scale such as $T(\text{anonymous})=0$, $T(\text{password})=1$, $T(\text{signature})=2$, $T(\text{fingerprint})=3$, $T(\text{iris})=4$, etc., and within a technology genre individual products might also be relatively ordered, such as $T(\text{Privaris fingerprint})=3.2$ and $T(\text{iGuard fingerprint})=3.5$.

Setting the trust levels for differing authentication technologies in the local trust domain is straightforward and no more difficult than current practice. Systems administrators already make decisions about which technologies they trust, and how much. However, if the authentication extends beyond the local trust domain, then we need trust authorities that can mediate the assignment of trust levels; for that, *federation* is required as discussed in section 6.

5. Authorization

In the conventional role-based access control (RBAC) model, a typical authorization policy is represented as “User U in role R has permission P.” However, to make our access control infrastructure aware of context information, it is necessary to define context-related constraints in authorization policies. We permit access policies such as “User U in role R who satisfies constraint C has permission P.” The depiction of this type of policy is shown in figure 2.

Here, a constraint is defined as a restriction that can be applied by the authorization policy: permission P is granted to role R if and only if constraint C is satisfied. Numerous types of contexts are possible, but we are mainly concerned with the context of the current access request (e.g., the status of the user making a request; the status of the object being requested; when and where the request originated). By adding context-based constraints to the authorization policy, authorization can be determined dynamically based upon the current context of the request, rather than just the role of the user.

A constraint is composed of multiple restrictions, called *context conditions*. A context condition is defined as a predicate on some context type, the value of which determines whether a particular condition is satisfied. The context condition is normally stated as a Boolean expression (e.g., $\text{time} > 14:30$, $\text{IPAddress} = 128.45.18.3$). To simplify that expression we introduce two new terms: *context type* and *context implementation*.

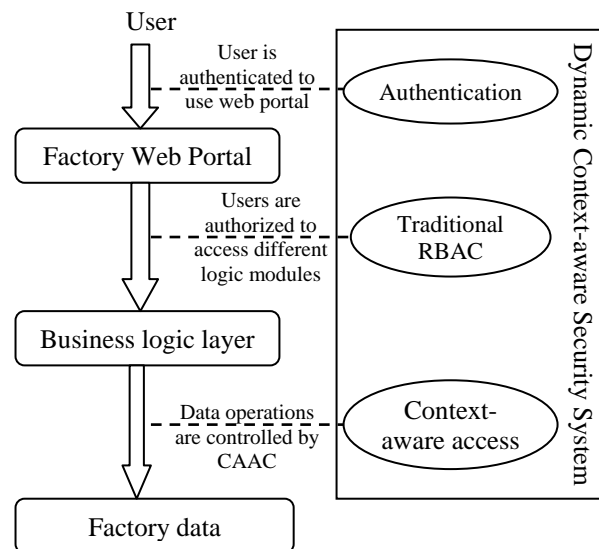


Figure 2. Authorization Logic

A *context type* is defined to be a property of the situation under which the access request is issued. In simple cases, context type may be a concrete property familiar in everyday life, such as time or location. In a more complex scenario, context type can also be used to describe an abstract concept such as authentication trust level; this would allow us to add a context constraint such as “authentication-trust-level \geq T(password).”

The *context implementation* of a context type is defined as a program or function that can evaluate the value of the context type. A context implementation can be a local function call, a program execution, or some remote service as long as it can determine the value of the current context type. Because context type can be an abstraction, there is no limitation on how it can be evaluated. For example, for a context type of *time*, it could be evaluated using a local time function, or by using a standard time service on a remote server, or by invoking a web service. Computing the value of a context type is called *context evaluation*.

The process to determine whether an object access is authorized is:

- 1) Retrieve all access control policies that should be applied to the requested object.
- 2) For each policy, extract all constraint information associated with it.
- 3) For each context condition in the constraint, first evaluate each context type via its implementation, then return that value to the predicate, and finally evaluate the context condition.
- 4) Compute the Boolean result of all context conditions to determine whether the access policy is satisfied.
- 5) If the access policy is satisfied, object access is authorized, otherwise it is denied.

6. Federation

Federation is a collection of realms or domains that have established trust. As a real-life example, consider the case of using one bank’s debit card in another bank’s ATM – the networking and security infrastructure determine whether the identity established at bank A is sufficiently reliable for acceptance at bank B. Federated systems can operate across organizational and technical boundaries, including different operating systems and different security platforms.

Federation depends upon two authorities being resident in each domain:

- Security Token Service (STS) – a web service that issues security tokens; it makes assertions to whoever trusts the STS based upon evidence that the STS itself trusts.
- Identity Provider (IP) – this entity acts as an authentication service to end requestors, and is an extension of a basic STS service.

Because trust domains are independently established and maintained, federation must address different trust topologies; it must model existing business practices and at the same time leverage existing infrastructure. As an example, suppose that a user has legitimately established his identity and received an authentication token within the factory trust domain. If the user now wishes to access data at a different but cooperating manufacturing facility, how can the trust established in the factory trust domain be exported to the manufacturing trust domain?

As a work-in-progress, we are investigating three solutions:

- *Token exchange*. Alternatively, if system A wanted to make a service request of system B, and if system A’s STS trusted system B’s STS, then system B could issue an access token valid in system B (an exchange token) based upon its trust of the system A STS’s assertion that identity had been satisfactorily established within system A.

- *Token validation*. System A makes a service request of system B. If the two STSs trust each other, then the system A STS can provide the local identity token that it created and certify its validity; the system B STS can then certify the local identify token from system A for use in system B.

- *Indirect trust*. The more general solution would borrow from the schemes used for validating digital signatures using certification authorities. In this case, system A’s STS trusts a known set of other STSs, and system B’s STS trusts a known set of STSs. If the trust set of system A has a member in common with the trust set of system B, then that common member in the two trust groups can validate the security tokens of each system to the other. The common member’s STS then certifies identity tokens originating in system A to STS B and vice versa.

Another consideration is that the trust level definitions of one domain may not be consistent with those of a separate and independent domain. For that reason we provide a mapping function that allows a system administrator to map the trust levels of one domain into those of another.

Acknowledgement

I gratefully acknowledge the research contributions of my students James Hu, Xiaohui Chen, James Van Dyke, Andrew Marshall, Vincent Noël, Xiudian Fang, and Zhengping Wu, my research collaborator Dr. Samuel J. Dwyer III, and our research group’s generous financial support from Mr. David Ladd at Microsoft Research.