A Security Architecture for Data Privacy and Security

Alfred C. Weaver
University of Virginia
Department of Computer Science
Charlottesville, VA 22904 U.S.A.

Abstract

Data access and software exchange are often achieved over insecure networks such as the public Internet. System designers are therefore forced to be proactive with regard to verifying the identity of both human users and software processes that request access to protected resources such as factory data. In this paper we show a new security architecture based upon web services that supports authentication, authorization, and federation. Authentication verifies identity and generates a security token; authorization determines which privileges are allowed to which users; federation permits secure and reliable exchanges of identity across disparate trust domains. We illustrate how these ideas can be used to secure access to a factory web portal and its underlying database of process data.

1. Introduction

A recurring concern with intelligent, distributed systems is the privacy and security of process data and software exchanges. As factory monitoring and control migrate from proprietary, closed systems to open, intranet- and internet-based systems, the risk of information leakage, malicious invasion by hackers, and damage due to software viruses increases to unacceptable levels unless system designers are proactive with regard to security. We propose a security architecture based upon web services that provides information security for factory data exchange initiated by both human users and software processes. The key steps are authentication, authorization, and federation, and are applicable to both human users and software that request access to any protected resource. Authentication results in a security token that conveys both the identity of the requestor and the trust level of the identification technology. Authorization determines what objects are accessible by a user given his identity token, request, role, context, and access privileges. Federation, using both direct and indirect trust, addresses the problem of how identity, once legitimately established in one trust domain, can be reliably exported to another cooperating trust domain. In this paper we describe each of these aspects in turn.

2. Security Architecture

Figure 1 illustrates how a wireless PDA might be used to access a 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 view the requested data? Is he allowed to change process control parameters? What if the data request originates with a software process rather than a human user?

Following the web services architecture recommended by IBM and Microsoft in [1], all data requests are made through the agency of the *factory portal*, a common entry point for all accesses to protected resources. A WS-Policy document defines what authentication tokens are acceptable as proof of identity for login. Upon initial access, a user is redirected to the authentication web service to establish identity and generate an authentication token; the token is stored on the access device as a cookie, 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. Software applications running on the portal attempting to access data use digital signatures to authenticate their origin.

After successful authentication and issuance of the security token, all portal data requests are sent to the factory data web service along with the user's authentication token. The portal's WS-Policy document defines the allowable or required authentication tokens that must be included with all data requests, while the factory data web service's WS-Policy document defines both the authentication and authorization requirements for access to protected resources (e.g., data). All WS-Policy documents are XML-based.

If the data service's WS-Policy is simple, it can be enforced automatically using primitives provided in Microsoft's Web Service Enhancements (WSE [2]) package; if the policies are complex, then WSE can call an authorization web service that supports custom policy assertions. The authorization engine in turn consults its

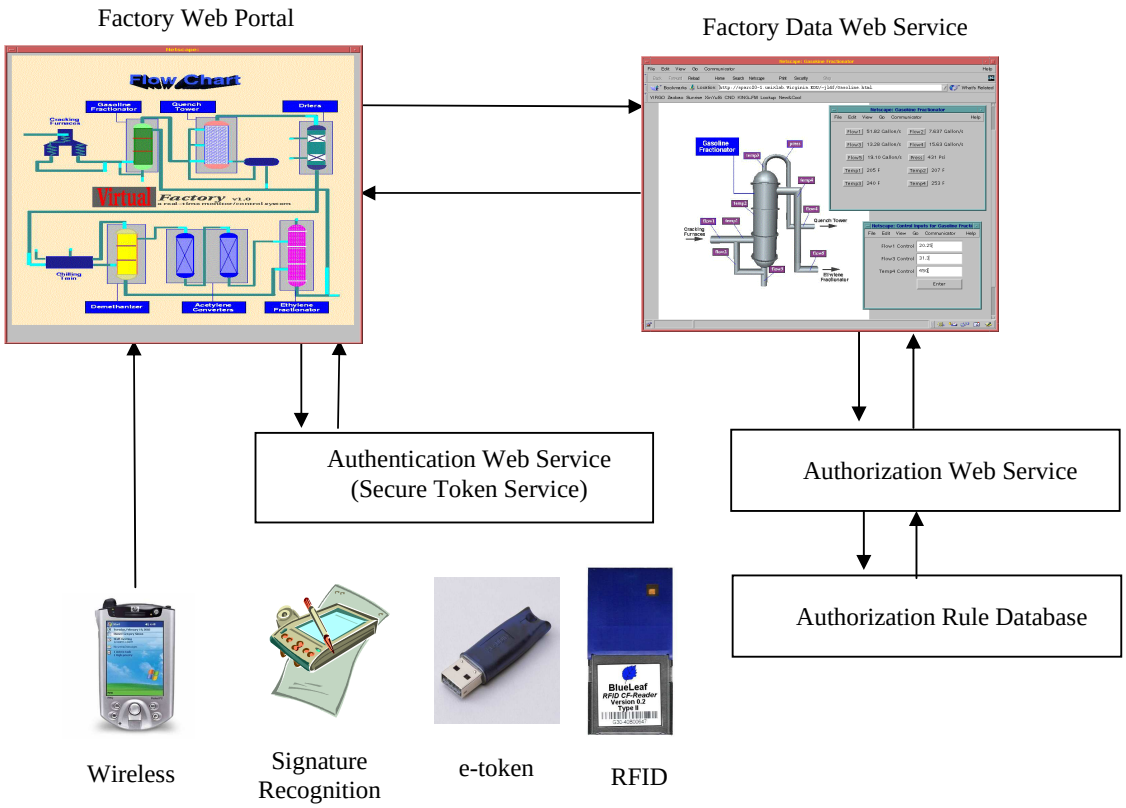

Figure 1. Security Architecture

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 directly or by a customized 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. Context allows us to evaluate local issues. For example, did this data request originate from a wired or wireless device? Did it originate within or without the local trust domain?

We recognize that not all data access requests originate from within the factory's trust domain; manufacturing partners and other allied services may have a legitimate need to access or even modify process data. For that reason we incorporate the concept of *federation*, which is a method whereby identity, once legitimately established in one domain, can be reliably and securely exported to another trust domain.

3. Authentication

Authentication of software processes is achieved using digital signatures. Authentication of human users is achieved by having the user present a previously created authentication token at the time of login. If the authentication token has expired, or if this is the user's first login, then the token can be (re)created by the Secure Token Service (STS) associated with the authentication web service.

Currently we support legacy authentication techniques such as passwords, biometric techniques such as fingerprints, iris scans, and signature recognition, digital techniques such as e-tokens and RFID, plus a two-factor technique that requires both a password and a random number that changes once per minute. For all of these techniques the result is the same—the STS issues an authentication token that identifies the user, the name and URL of the authentication service, the expiration time of the token, and the token's *trust level*—a numerical representation of the level of trust that the system administrator assigns to the identification technology used.

The resulting authentication token (security token) looks like this:

```
<TrustLevelSecToken>
  <CreatedAt> 05/28/2005 08:00:00  </CreatedAt>
  <ExpiresAt> 05/28/2005 17:00:00 </ExpiresAt>
  <UserID> 83245797 </UserID>
  <TrustLevel> Fingerprint </TrustLevel>
  <TokenIssuer> http://cs.virginia.edu/TrustSTS.asmx
  </TokenIssuer>
  <TrustAuthority>
    http://cs.virginia.edu/TrustAuthority.asmx
  </TrustAuthority>
</TrustLevelSecToken>
```

Figure 2. Authentication Token

In principle, a biometric technique such as an iris scan is a more reliable identification technology than a fingerprint, which is in turn more reliable than a password. By allowing the administrator to create a reliability ordering among these devices, that trust level can then be incorporated into authorization rules, such as "require authentication at the level of a fingerprint or higher," or requiring a higher level of authentication security in the case of a wireless access as opposed to a wired access from within the trust domain. In practice, the relative ordering of the reliability of various identification methods and devices is determined by experience and values are assigned by the system administrator.

4. Authorization

In the conventional role-based access control (RBAC) model of Sandhu et al. [3], a typical authorization policy is represented as "User U in role R has permission P." We have extended RBAC to include local context information as suggested by Covington et al. [4] and by Neumann and Strembeck [5]. 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 3.

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, for example the status of the user making a request, or the status of the object being requested, or 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

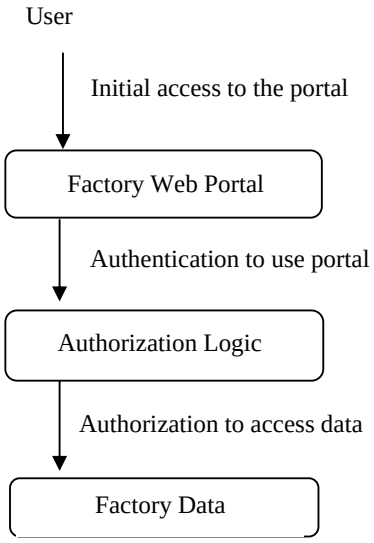

Figure 3. Authentication and Authorization Logic

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., time > 14:30, IPAddress = 128.45.18.3). To simplify that expression we introduce two new terms: *context type* and *context implementation*.

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 T(x); this would allow us to add a context constraint such as "authentication trust level >= 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*.

Based upon the context type and its implementation, 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.

This approach implements the *principle of least privilege* as advocated by Schneider [6].

5. 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 wherein the networking and security infrastructure of the banks 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?

We are investigating two different techniques for trust establishment: direct trust and indirect trust. *Direct trust* can be used when the number of cooperating domains is modest and can be managed administratively. Using this technique, two cooperating domains establish a business relationship with each other and the administrators of each domain write a WS-Policy that explicitly allows each domain to accept the security tokens issued by the other domain. After validation, the external security token may be exchanged for an internal security token that is valid in the local domain. Indirect trust can be used when the number of cooperating entities is too large for manual administration and automatic techniques must be employed. Using this method, when domain A seeks to begin a trust relationship with domain B, but they have no pre-established relationship, then A and B must search their respective lists of trusted domains (their list of trusted Secure Token Servers in other domains) to find one, say C, that is trusted by both domains. Then, because A and B both trust C, C can validate the security tokens of both A and B and issue a new token that both A and B can trust. Upon receiving the trusted token from C, both A and B can exchange it for a token that is valid in the local domain.

6. Work-in-Progress

Our proposed security architecture was first developed for protecting the privacy and security of medical data [7]. We have implemented a prototype of the system described above and have experimented with various authentication methods and devices, authorization policies and representations, and trust brokering among cooperating trust domains using both direct and indirect trust strategies.

7. Acknowledgement

This security architecture is the work of many people, including my graduate students Zhengping Wu, Brian Garback, Paul Bui, Joe Calandrino, and Jim Van Dyke. All of us are indebted to our very able assistant, Kim Gregg.

References

[1] IBM and Microsoft, "Security in a Web Services World: A Proposed Architecture and Roadmap," http://msdn.microsoft.com/library/en-us/dnwssecur/html/securitywhitepaper.asp, April 2002.

[2] Web Services Enhancements (WSE) version 3.0, http://msdn.microsoft.com/webservices/building/wse/default.aspx

[3] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman, "Role Based Access Control Models," IEEE Computer, vol. 2, Feb. 1996, pp. 38-47.

[4] Michael J. Covington, Wende Long, and Srividhya Srinivasan, "Secure Context-Aware Applications Using Environment Roles," Proc. Sixth ACM Symposium on Access Control Models and Technologies, May 2001.

[5] Gustaf Neumann and Mark Strembeck, "An Approach to Engineer and Enforce Context Constraints in an RBAC Environment," Proc. Eighth ACM Symposium on Access Control Models and Technologies, June 2003.

[6] Fred B. Schneider, "Least Privilege and More," IEEE Security and Privacy, Sept/Oct 2003, 1(3), pp. 55-59.

[7] Jenzhe Hu and Alfred C. Weaver, "Dynamic, Context-Aware Access Control for Distributed Healthcare Applications," Workshop on Privacy, Security, and Trust, August 2004.