

Comparing the security architectures of Sun ONE and Microsoft .NET

Eduardo B. Fernandez, Michael Thomsen, and Minjie H. Fernandez
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431, USA

Abstract

Platforms for web services have been reduced to two basic approaches: Microsoft .NET and Sun ONE (J2EE). We compare here these two platforms with respect to the security they provide to the web services that use them. We arrive to the conclusion that although the basic security architectures are fairly similar, their actual implementations differ. Microsoft's approach appears weaker because of their self-contained approach and not following good principles of software and secure systems design.

INTRODUCTION

Several companies have announced strategies for supporting web services. They all use one of two basic architectures: Microsoft's .NET or Sun ONE. Microsoft's architecture is also an implementation, while Sun ONE is an architecture with several implementations based on Java. Comparisons of these architectures barely mention security [Mou01, Ses01]. We believe this aspect is one of the most fundamental factors in their success or failure and we look here at the security features in Microsoft .NET and Sun ONE web services architectures.

The basic purposes of an architecture to support web services are:

- Store the web service programs on shared systems, where users can find and access them.
- Some systems also need to store the user's data on these shared systems.
- Some systems store web services repositories or catalogs.

This brings up the typical security issues, which now have the following objectives:

- Confidentiality: The web service data may only be accessed by authorized users.
- Integrity: Web services data, code, or descriptions may only be altered by authorized users.
- Code control: The program code must not be able to perform illegal actions when invoked.
- Access control: Only paying subscribers can use the service.
- Availability: Code and data stored on the server must be available when needed.

Some of these issues should be resolved in the upper layers, where web services are defined. These layers are based on standards under development and we do not discuss these aspects here because they apply to both architectures. A survey of security aspects of web services is given in [Fer02].

Because both the Microsoft .NET and Sun ONE architectures are quite new, there are many aspects still not tested in practice. These two platforms are still evolving and some of our specific conclusions may not be true anymore, but unless there is a change in their fundamental design our main conclusions should still hold.

The next section provides a general overview of both architectures. We then consider in detail the security architectures of Sun ONE and .NET, relating them to the standard work on security models. We end with a general discussion of both approaches.

A GENERAL COMPARISON OF MICROSOFT .NET AND SUN ONE

Both the Sun and the Microsoft frameworks are based on specific programming languages, object models, and virtual machines, but the design approaches of their runtime environments are quite different. We summarize here a few aspects, several papers compare them in detail [Far01, Mou01, Ses01, Vaw01].

Sun ONE is based on Sun's J2EE (Java 2 Enterprise Edition) specification, which implies that you'll have to use Java as programming language, but the programs can (in theory) run on any platform without recompiling. The Microsoft .NET framework builds on Microsoft's Common Object Model (COM), that has been improved significantly, including additions such as ActiveX, distributed computing, database connections, etc. Microsoft has also produced a new programming language, C# and a new runtime environment called Common Language Runtime (CLR), that provides garbage collection and some security features similar to the Java Virtual Machine (JVM).

J2EE accesses data sources using the JDBC API, which requires vendor specific drivers. These are available for most commercial and open-source databases. Another approach to persistent objects is Java Data Objects (JDO) [jdo]. Java uses JNDI (Java Naming and Directory Interface) in order to access directory services. Support for various directory services, such as Novell NDS, and Directory Services Markup Language (DSML), among others, is available. Some XML data sources use drivers that are similar to the JDBC drivers.

Microsoft's first data access service was DAO (Data Access Objects), that later led to ActiveX Data Objects (ADO) which provides access to both relational data and directory services. The .NET framework with ADO.NET improves on some of the limitations of ADO (e.g., lack of scalability). Another change is that the data transmission format has been replaced by XML, which gives the opportunity for components to act as data sources for each other.

Although Sun ONE is based on Java, which is said to be platform independent, some implementations are not exactly cross-platform independent. On his part, Microsoft has made some attempts to make the .NET framework platform independent, such as submitting the CLR specification and the C# language for standardization, but more realistically, Sun ONE is dependent on Java, and Microsoft .NET incorporates Windows as a central unit. Furthermore the .NET language independence is somewhat incomplete, since you'll have to add some extensions to the languages in order to make them comply with the CLR requirements, thereby making them COBOL#, Eiffel#, Perl#, etc.

Microsoft has been very active in the XML field for the past few years, and .NET provides strong XML support. As mentioned earlier, XML is used for data transmission, but also for general-purpose document handling. Web services can be exported as SOAP interfaces (Simple Object Access Protocol). Sun has also been active in the XML field, and XML has been incorporated into their framework. Table 1 shows a basic comparison of these approaches.

SUN ONE SECURITY ARCHITECTURE

Overview

Let's take a look at a service request. A request for a protected resource goes through standard Internet protocols, such as HTTP (Figure 1).

	Microsoft .NET	Sun ONE
Programming Language	Any (but extensions needed)	Java
Runtime environment	CLR, code is compiled into native code before execution	JVM, Java bytecodes, either interpreted or compiled on-demand
Component sharing	IL, CLR	JVM with Corba and ORB
Data interchange protocol	XML, SOAP on HTTP	XML, SOAP on HTTP

Table 1. .NET and ONE side by side

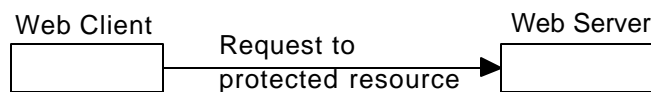


Figure 1. A service request

The web server detects that the new request has not yet been authenticated so the appropriate authentication mechanism for the resource is invoked. This may be done by returning a form where the user can fill out a user name and a password, but there are several other options (see below). The data is transferred to the web server that validates the data, and sets the credential for the user (Figure 2).

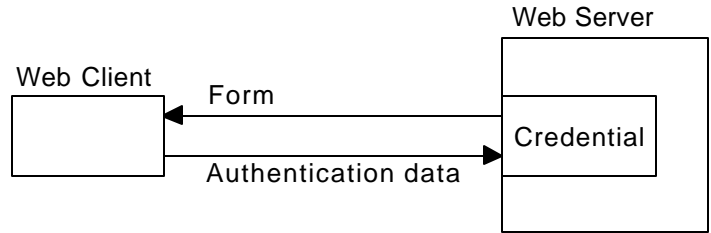


Figure 2 Client authentication

Once the web server decides that the request is authorized, it consults the authorization rules (called *security policies* by Sun) associated with the resource. Next, the web server container tests the user's credentials against each role in the authorization rules. This process will either stop with an "authorized" or a "not authorized" outcome depending of the web container being able to map the user to any of the permitted roles. If the user is authorized, the web server returns the result of the original request (Figure 3).

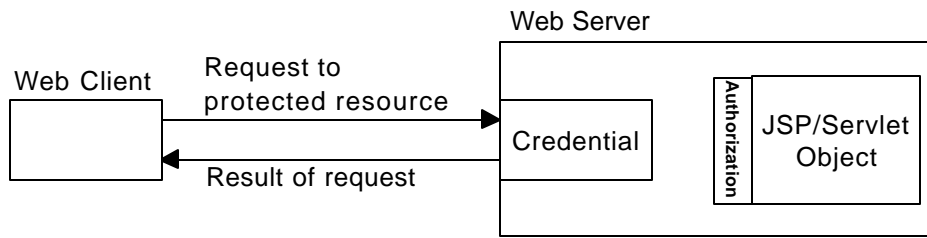


Figure 3 Result of authorized request

Later on, the user might perform some action that needs to be handled by an EJB component. When a JSP page performs a remote call to the EJB component the user's credential is used to establish a secure association between the JSP page and the EJB component (Figure 4). At this point it is the responsibility of the EJB containers to enforce the access control on their bean methods. This is done by consulting the authorization rules associated with the bean, and an "is authorized" or "not authorized" outcome is generated. EJBs may access databases which may enforce additional security constraints, e.g., content-dependent access control, where access depends on specific rules in the databases.

The following sections describe in detail these actions.

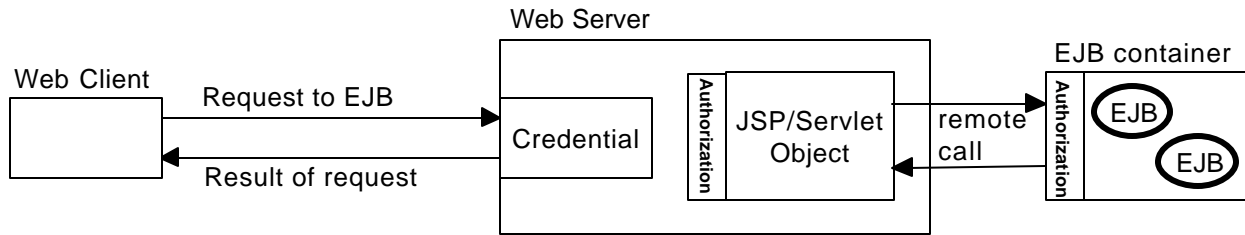


Figure 4 Application object access

Authorization

Security in J2EE is based on roles, they use a type of Role-Based Access Control (RBAC) model. In an RBAC model users are assigned to roles and roles are given rights for resources based on their functions, e.g., administrator, manager, developer. RBAC corresponds to a specialization of the access matrix model [Sum97]. In the access matrix model subjects (requesting entities) are authorized to access specific protection objects in specific ways (access types or access modes). The set of authorization rules defines the subjects' privileges and implements the institution security policies.

One of Sun's design goals when they created the J2EE platform was to separate the security aspects of the development of components, the assembly of these components into applications, and the deployment of these applications. Several roles have been defined: the *Component Provider* and *Application Assembler* specify which parts of an application require security, and the *Deployer* selects the specific security mechanisms used for that protection. This security is specified, among other information, in the deployment descriptor that is written in XML [Kov01].

J2EE introduced the JAAS (Java Authentication and Authorization Service), to enforce server side security. Enforcement of security is based on two approaches: declarative and programmatic security. *Declarative security* is based on authorization rules defined in a J2EE system by the deployment descriptor. This is a contract between the Application Component provider and the Deployer. Groups of components can be associated with one deployment descriptor. When declarative security is not sufficient to express the security constraints of the application, the *programmatic security* approach can be used instead. It consists of four methods, which allow the components to make decisions based on the security role of the caller.

The trade-offs between the declarative and programmatic security approaches are: The former is more flexible after the application has been written, and usually more comprehensible, and therefore results in fewer bugs. More important, it is done in a uniform way across the system. The latter could provide more functionality when the application is being written, but is buried in the application and is therefore difficult to change, and usually only fully understood by those who developed the application. Programmatic security does not allow checking for compliance with institution policies and alone is not acceptable for systems that require a high level of security [Sum97]. However, it could be useful to complement declarative security by adding special restrictions for application access, although it doesn't appear that it can be combined with declarative security in J2EE.

We have already mentioned that the J2EE authorization is based on roles. These roles are defined by the Application Component Provider, and are used by both the declarative and programmatic security approaches. If necessary each method in each bean can be assigned different permissions, i.e., this is a type of method-based access control [Fer94]. The Component Provider specifies what roles are being used by the application, a list of external resources accessed by the components, and references to all inter-component calls. A method permission model and information that identifies the sensitivity with respect to privacy of the information exchanged may also be provided. When the Application Assembler combines the components into an application he has to create a consistent security view for the application as a whole. Finally, the Deployer is responsible for securing the application in the specific operational environment. The Application Assembler should supply the Deployer with information on which method parameters or return values require protection; this information is added to the deployment descriptor. It would be nice if the deployment tools included message security. This would allow the application assembler to decide which channels require integrity and/or confidentiality checks. Later, tools could be used by the deployer to choose the best algorithm to solve the problem. It is important to note that the Component and Application Developers define the roles and the Deployer assigns users to the different roles. Several tools are currently available to help the Deployer configure the security mechanisms. These security mechanisms are implemented by the containers on behalf of the components hosted by the containers. As indicated earlier, J2EE components can access persistent data stored in relational databases. It is important that any authorization defined in a component is consistent with authorizations defined in the database system.

In the J2EE model, the deployment descriptor defines the security roles and associates these roles with the components in order to define the granted permissions. If a JSP/Servlet resource is not associated to any roles, permission to access the resource will be granted to all. This is risky business, and doesn't comply with the principles of closed systems and least privilege, fundamental for secure systems [Sum97]. The EJB components must, on the other hand, have associated at least one role to them. If unrestricted access is needed to an EJB component you'll have to map a role to the component which is permitted access to the resource without authentication. This way you cannot leave a component exposed to everyone because you forgot to assign a role to it.

Other authorization problems can occur if method level access control is applied to a component, because a less protected method could be used to undermine the policy enforced by a more rigorously protected method. For example, a method that allows to read a value directly. It is therefore recommended to partition the components that need different authorization rules, so that all methods of each component enforce the same guidelines. Actually, if one applies properly the RBAC model, defining roles from use cases [Fer97], this should not happen. Each use case defines the least amount of privilege needed by each actor to perform its job. Finally, when using XML documents, these may have their own authorizations, defined in terms of XACML or similar models. Mappings between these levels are needed to provide a consistent enforcement of the defined authorizations [Fer99].

Authentication and other aspects

The J2EE specification addresses two different client types for authentication, namely a web client, and an application client. We will focus only on the web client in this paper. There are three possible mechanisms to authenticate a web client:

HTTP Basic Authentication--This is the an insecure authentication method, since the passwords are encoded using base64 encoding.

HTTPS Client Authentication--This is a strong authentication mechanism that allows mutual authentication. It is based on X.509 certificates and requires the user to possess such a certificate. The authentication occurs over a channel that is protected by SSL.

Form-based Authentication--Lets developers customize the authentication user interface presented by the web browser using standard HTML or JSP/Servlet based forms.

It is recommended by Sun to perform client authentication only when the client tries to access a protected resource. This is a type of lazy authentication, and is convenient for the user. When a user has identified himself, the login session will be maintained so that the user can access multiple applications for which the user has the necessary rights. The session state is stored on the server, and the client keeps references of the state either by URL rewriting or cookies. It is possible for the client to manage its own authentication context without using cookies if SSL is used. In a secure system, all resources should be explicitly protected, so lazy authorization is not a good approach in general

EJB components need to be authenticated when they communicate with the enterprise data logic (e.g. an ERP system or a database) that is usually located in another security domain. The J2EE Connector Architecture Specification describes this issue in further detail. The following two mechanisms are required for the J2EE implementation, whereas the latter three are only recommended:

Configured Identity--The principal and authentication data are specified at deployment time, and are therefore independent of the principal of the caller.

Programmatic Authentication--The principal and authentication data are specified at runtime using APIs appropriate to the resource. The data can be obtained from the components environment or as parameters.

Principal Mapping--The resource principal is determined by mapping data from the security attributes from the calling principal.

Caller Impersonation--In this scenario the resource principal acts on behalf of the caller's principal. This requires that the caller's identity and credentials are delegated to the underlying resource manager (e.g. an ERP system or a database).

Credentials Mapping--This is used when the component and resource support different authentication domains. An example could be that a certificate used to identify the user could be mapped to some credentials for the resource.

Sun has joined the Liberty Alliance project [Lib02], a federated identity infrastructure, with a first phase that intends a web-based Single-Sign-On (SSO).

One way to ensure message integrity is to attach a signature to the message. By including a time-stamp and a serial number you can further ensure that the message can't be resent by a third party. Finally, encryption can prevent anybody from reading the message - provided that the encryption mechanism is strong enough. It is the deployer's responsibility to define the necessary security measures used, and it is the container's responsibility to enforce them. The way to do this is by appending the signature to outgoing messages and verifying signatures and timestamps on incoming messages, as well as notification of the caller in case of failure.

The Sun ONE specification describes some facilities for auditing [Sun02]. It is, however, the Deployer who is responsible for configuring the security mechanisms that are applied to the enterprise containers, and it is therefore also the Deployer's responsibility to apply proper auditing. Some general guidelines about what to audit and how to protect the audit data are mentioned in [Sun02].

As indicated, Sun ONE is only an architecture and there may be different implementations of it. Effective security will strongly depend on the specific underlying platforms, e.g., the web HTTP server, the application server, the operating system, and even the hardware. Sun's ONE server and IBM's WebSphere have good security reputations as application servers. They both use Apache as HTTP server, another strong component. Sun uses Solaris and IBM uses a variation of Unix (AiX) or Linux, which are reasonably strong operating system architectures. In other words, the security platforms normally used for Sun ONE appear strong based on their general architectures and their track record.

SECURITY IN MICROSOFT .NET

Several security functions in .NET are similar to their counterparts in Sun ONE and we don't repeat them here. We concentrate instead on those aspects which are different. Further details of the .Net security architecture can be found in [LaM02].

As indicated earlier, the web services provided by Microsoft .NET are accessed over the Internet through HTTP, SOAP, and XML. The Common Language Runtime (CLR) implements the new code access security, including controlling that code cannot perform unauthorized actions. The security policy is managed and the security checks are performed when the code is loaded and executed. User access control is managed by the web server or the operating system that controls the server (Figure 5).

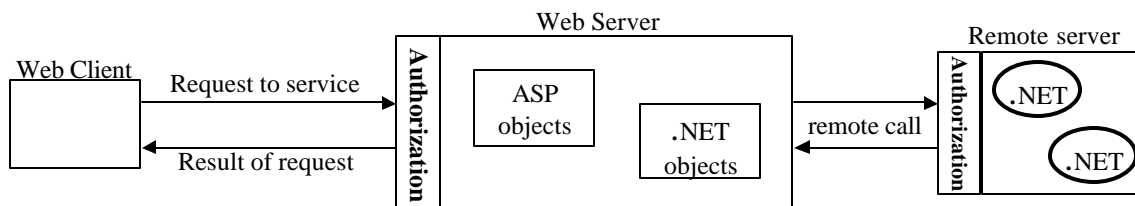


Figure5 Accessing objects in .NET

The code access security is used by the programmer to specify which permissions their code requires, and for the user to secure his system from malicious code. These permission requirements are stored in the components at the assembly level. Also programmatic security checks can be coded into the components if necessary, and as in the EJB components it is possible to have method-level access control.

When the code is loaded, the runtime library verifies that the code can be granted the permissions it has requested. Policies for granting permissions on the client are established by system administrators. One interesting feature is that the component and the security manager can negotiate the security level. For instance if a component wants to use file I/O, but can execute without it, it will request file I/O permissions. If these permissions are denied, the component can still execute, but without accessing files. This appears dangerous from a security perspective, a rogue program could try to access different resources until it succeeds.

User access also uses RBAC. Roles can be defined at development or deployment time. At runtime the identity of the user on whose behalf the code is running is determined, and access is granted or denied based on those roles. These roles are typically mapped to credentials in Microsoft's Active Directory.

Role access in .NET components can be inherited from process to component to interface to method. This is an example of Implied Authorization [Fer75], where access to a composite implies access to components; in particular, this can be interpreted as inheritance of authorizations along a generalization hierarchy [Fer94]. In .NET higher-level-defined accesses override lower-level rights [Low01], an opposite policy to the one defined in [Fer94]. For example, I may want to give access to the whole Student class to somebody but not allow that person to see the grades of specific students, something that cannot be done in .NET. A lower-level access is a more precise (finer) specification and it should override the higher and coarser access.

If you know exactly which computers should be allowed access to the application then the Internet Protocol Security (IPSec) or a firewall can be used to restrict access. Since this is not possible in most scenarios Microsoft suggests that you move this authorization to the protocol used to exchange messages. If you're sending and receiving messages using SOAP over HTTP you can use the authentication features available for HTTP. Microsoft's Internet Information Services (IIS) provide several authentication mechanisms for HTTP. This includes support for HTTP Basic, HTTP Basic over SSL, and Client Certificates. An additional authentication mechanism is the *Integrated Windows authentication*, which uses a proprietary password-hashing algorithms and requires both client and server to be Windows systems. Windows authentication is based on Kerberos, but cannot be used over a proxy server or a firewall. IIS 6.0 will also include Passport as authentication approach, although there are doubts about its security [Opp03].

Another authentication method is based on custom security mechanisms built by the developer. It is possible to use the SOAP body to communicate the credentials. Since it is difficult to receive the data from the SOAP header, Microsoft does not recommend using the header for this purpose. One

drawback of using the SOAP body is that you'll have to make the debugging yourself. Another drawback is that the SOAP messages are sent over HTTP, hence all data is sent in cleartext. So if this technique is used you should definitely use SSL to encrypt the messages between the client and server. The main issue about using SSL is that it is a lot slower than HTTP itself. Therefore Microsoft suggests that you only encrypt part of the message body, and use digital signatures to ensure that the body has not been tampered with in transit.

The .NET framework includes cryptographic functions for encryption, digital signatures, and hashing. The supported algorithms includes: RSA and DSA for asymmetric encryption, and AES, DES, Triple DES, and RC2 for symmetric encryption. The implementation uses a stream-based model suitable for networking [MS01]. The XML Digital Signature Specification (XMLDSIG) is now also supported. This protocol can be used in combination with SOAP to ensure the integrity of the messages. This applies also to Sun ONE since they are following the same standards.

The current .NET framework specification does not say anything about auditing. We assume Microsoft expects you to use the auditing features included in Windows 2000, or to code your own.

Microsoft's server IIS has shown many weaknesses because of its complexity and poor design. This makes it probably the weakest component in the architecture now. Windows 2000 also appears to have a variety of security problems, in particular, in its Active Directory. Security improvements in IIS 6.0 may correct these problems, we have to wait and see.

CONCLUSIONS

It turns out that the two platforms are remarkably similar when it comes to their general security architectures. Both use RBAC as authorization model, both support "deployment descriptors", method level access control on the component code, and declarative as well as programmatic access control to the components. Finally the two frameworks include an extended sandbox model for the executing code. In both systems, roles do not enforce content-dependent authorization, this aspect is delegated to the DBMS. An interesting feature of Microsoft is inheritance of rights, although this idea is not used in the most secure way.

Both frameworks also lack some security features. Sun's JSP objects are left unprotected if the deployer forgets to assign a role to them (open system policy). Fortunately, the EJB components do not suffer from that, so the situation can be controlled. Finally, auditing is not clearly integrated with the architectures, Sun relies on the J2SE logging facilities while Microsoft delegates this function to the operating system. It would have been nice if both had chosen to use XML for writing the auditing log. This would allow a lot of log-tracking and statistical programs to work with both frameworks without much additional coding.

Their main differences are in their approach to authentication, their lower levels, and their general approach for developing software. For authentication Sun follows the norms of the Liberty Alliance while Microsoft follows its own path. It is not clear now which approach is stronger but the insistence of Microsoft in using its own standards is not good for security. With respect to their lower levels, one

needs to weigh Windows against Solaris or AiX as secure platforms. Solaris and AiX appear to have the edge here, although the security architecture of Windows appears well thought. A larger difference is in their approach to software development (Most of these defects are also present in other vendors, but to a lesser extent):

- No use of object-oriented programming. Most (or all) of their systems code has been done using procedural methods. Without proper software design, it is almost impossible to build dependable complex systems.
- Lack of modularity. As a marketing strategy, Microsoft builds systems that cannot be decomposed, i.e., they lack well-defined interfaces. This prevents checks across the interfaces and a vulnerability in one place propagates its effects to other places.
- No use of secure system design principles. These were formulated as early as 1975 [Sal75] and include aspects such as open design, fail-safe defaults, and similar. Microsoft's approach to develop secure code is to look for specific vulnerabilities in the code, a very difficult task.

It is important to mention again that Sun ONE is an architectural specification and not a product. Anyone can create an implementation of the Sun ONE environment on any system. Sun has developed an implementation for their own Solaris operating system and Sun ONE server, while IBM, BEA, and Oracle, among others, have implemented (at least parts of) the specifications on other systems. This means that the effective level of security in each of these systems can be different from each other.

Another aspect, not quite clear yet, is how their security approaches match the security defined at the web services level [Fer02]. Standards for these levels are still being developed and there is no experience with their use.

REFERENCES

- [Far01] J. Farley, *Microsoft .NET vs. J2EE: How Do They Stack Up?*
http://java.oreilly.com/news/farley_0800.html
- [Far01a] J. Farley, *Picking a Winner: .NET vs. J2EE*,
<http://www.sdmagazine.com/articles/2001/0103/0103a/0103a.htm>, March 2001
- [Fer75] E. B. Fernandez, R. C. Summers and T. Lang, "Definition and evaluation of access rules in data management systems," *Proceedings 1st International Conference on Very Large Databases*, Boston, 1975, 268-285.
- [Fer94] E. B. Fernandez, E. Gudes, and H. Song, "A model for evaluation and administration of security in object-oriented databases", *IEEE Trans. on Knowledge and Database Eng.*, vol. 6, no. 2, April 1994, 275--292.
- [Fer97] E.B.Fernandez and J.C.Hawkins, "Determining role rights from use cases", *Procs. 2nd ACM Workshop on Role-Based Access Control*, November 1997, 121-125.

- [Fer99] E.B.Fernandez, "Coordination of security levels for Internet architectures", *Procs. 10th Intl. Workshop on Database and Expert Systems Applications*, 1999, 837-841.
- [Fer02] E.B.Fernandez, "Waltzing through port 80: Web services security", *Software Development*, September 2002, s4-s7 and s14-s15.
- [jdo] Java Data Objects, <http://access1.sun.com/jdo/>
- [Kov01] L. Koved, A. Nadalin, N.Nagarathan, M.Pistoia, and T. Schrader, "Security challenges for Enterprise Java in an e-business environment", *IBM Systems Journal*, Vol. 40, No 1, 2001, 130-152
- [Lam02] B.A. LaMacchia, S. Lange, M. Lyons, R. Martin, and K.T.Price, *.NET framework security*, Addison-Wesley 2002.
- [Lib02] Liberty Alliance Project, <http://www.projectliberty.org>
- [Low01] J. Lowy, *COM+ (and .NET) make security a joy*, talk at Software Development West, April 2001.
- [Mou01] P. Mougine and C. Barriolade, *Web Services, Business Objects and Component models*, October 2001,
http://www.orchestranetworks.com/en/docs/0105_whitepaper.cfm
- [MS01] Microsoft Corp., *About .NET security*, 2001,
http://www.gotdotnet.com/team/clr/about_security.aspx
- [Opp03] R. Opplinger, "Microsoft .NET Passport: A security analysis", *Computer*, July 2003, 29- 35.
- [Sal75] J. Saltzer and M.D.Schroeder, "The protection of information in computer systems", *Procs. of the IEEE*, Vol. 63, No9, 1278-1308. Also in:
<http://web.mit.edu/Saltzer/www/publications/protection/index.html>
- [Ses01] R. Sessions, *Java 2 Enterprise Edition (J2EE) versus the .NET platform: Two visions for eBusiness*", White paper, ObjectWatch, Inc., March 28, 2001,
<http://www.objectwatch.com>
- [Sum97] R.C.Summers, *Secure Computing: Threats and Safeguards*, McGraw-Hill, 1997.
- [Sun02] Sun ONE Architecture Guide, 2002,
<http://www.sun.com/software/sunone/docs/arch/index.html>

[Vaw01] C. Vawter and E.Roman, *J2EE vs. Microsoft .NET—A comparison of building XML-based web services*, June 2001.