Enterprise Application Security

Keys Botzum
Senior Consulting I/T Specialist
IBM AIM Services
keys@us.ibm.com
September 17, 2000

1. Introduction

This document outlines some basic issues related to securing enterprise applications in a way that leverages corporate resources across independent applications. The proposed techniques can lead to substantial efficiencies and improvements in overall security.

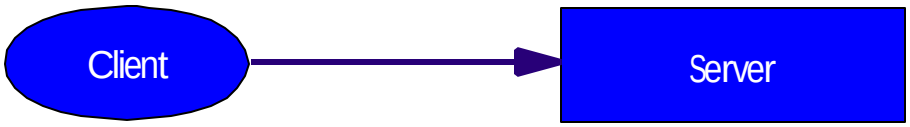
The motivation for this white paper is the securing of enterprise applications consisting of a large number of applications used in the same organization or company that share a common infrastructure. Sharing infrastructure has the potential to improve interoperability of applications, reduce costs, simplify user management, and improve overall system security.

The first section of this paper discusses a simple model of securing a distributed application in order to make some basic points. Then, the following few sections provide introductory material on Lightweight Directory Access Protocol (LDAP) and Public-Key Infrastructure (PKI) technologies. The use of these technologies in a practical security infrastructure is then discussed in sections 6 through 8. Beginning with section 9, a few important security topics are discussed: secure delegation, the use of passwords, and policy-based security tools. After the conclusion, the Appendices provide substantial supporting examples.

This document strongly advocates the use of the LDAP in conjunction with certificates as part of a Public-Key Infrastructure. Such an infrastructure is commercially based, widely supported, likely to succeed, and most importantly of all, practical. Simply providing certificates for authentication is not sufficient when one considers the real problem that is to be solved: securing applications and providing appropriate access controls to those applications.

Lastly, this paper is not intended to provide the reader with a complete understanding of any of the issues discussed. The problem of securing applications is large and complex. Interested readers should pursue the materials listed in the Reference section.

2. Security Model for a distributed application



This diagram seems laughably simple, but in fact, all distributed systems eventually break down into something like this. Two parties are communicating with each other. One party, the client, initiates the communication to the second party, the server. Although deceptively simple, the security implications of a distributed environment are not so simple. In order to secure this simple application, several things must be done:

1. There must be secure transport between the client and the server. The role of the secure transport depends on the application needs. Its requirements may include passing security tokens, protecting data from modification

(secure checksums), and data privacy (encryption). This document does not discuss the issues of secure transport.
2. The server must identify itself to the client.
3. The client checks that the server is some expected identity. For example, Web browsers ensure that the Web server is actually using an identity consistent with its DNS name.
4. The client must identify itself to the server. Typically, but not always, the client is acting on behalf of some human.
5. The server must take the identification information and validates it using some trusted technique.
6. The server must then use the client's known identity to obtain authorization information for the client from some registry. In essence, the server must learn what the client can do.
7. The server must enforce access to its resources based on the client's authorization information. The server will authorize the client as appropriate.

After a secure session is established, future client requests often execute only the last step as other information is cached.

These steps can also be viewed as two fundamental steps that are broken down into sub-steps. First, authentication consists of two steps: identification and validation. After authentication, comes the authorization step consisting of two steps: security lookup & rule enforcement. In more detail:

- **Authentication** (corresponds to steps 2 through 5)
  - *Identification* – one participant provides to the other some form of identification. This is some secret that the receiver can validate. Most commonly, it is a password or a PKI exchange.
  - *Validation* – the receiver takes the identification information and verifies it.
- **Authorization** (corresponds to 6 and 7)
  - *Security Lookup* – the receiver of a request uses the known identity of the caller to determine what access rights the caller possesses. Most commonly, the access rights are represented as a set of groups obtained from a file or enterprise directory such as an LDAP directory.
  - *Rule Enforcement* – the receiver uses the caller's access rights and compares them against some set of rules to determine if access should be allowed or denied based on the action requested. Most commonly, the rule is represented as an Access Control List (ACL). These rules are usually stored locally with the data they protect or in some application specific way. Rules can also be externalized via policy management tools.

This scenario does not address the issue of multi-tier authentication. When authentication spans tiers, the server must contact a second server (perhaps a database) on behalf of the client. The middle server may want to use delegation to send the client's credentials to the next server, thus allowing the next server to know the true client identity. In general, delegation is very difficult to implement and requires substantial infrastructure support. Additionally, it must be done outside of the usual basic public-key paradigm. WebSphere Application Server[1] supports secure delegation using a proprietary mechanism. For a more detailed discussion of delegation, see the later Delegation section.

Now that we have outlined some of the basic issues in a generic way, the remainder of this paper will go into more detail about currently available and widely supported standards for implementing enterprise security. Today, the most promising technologies for securing enterprise applications are LDAP and public-key certificates.

3. Basics of LDAP

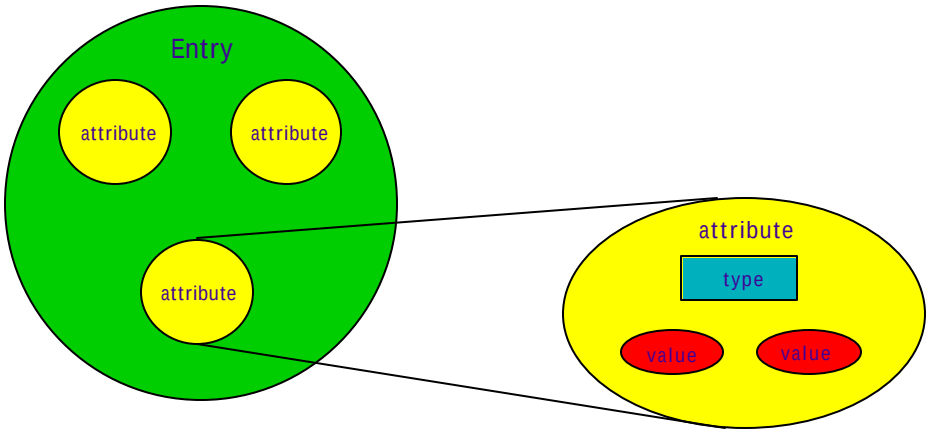
The Lightweight Directory Access Protocol (LDAP) is the preferred access method for accessing directories. A directory is nothing more than a repository of searchable information. In many ways a directory is like a database, but optimized for different purposes: queries far outnumber modifies, simple access methods, loosely coupled wide-area replication. LDAP is a set of standards for accessing a directory. LDAP is based on the X.500 DAP protocol and was originally intended to simplify it (mostly by not requiring OSI protocol suite). Additionally, LDAP

[1] Throughout this document WebSphere Application Server refers to WebSphere Application Server Standard and Advanced Editions, versions 3.0.2 and 3.5. Enterprise Edition is based on different technologies and is not covered here.

specifies only the client view of a directory. Server to server communication is not specified (as they are in the X.500 suite). Thus, an "LDAP directory" is really something that can be accessed using the LDAP client protocols. The implementation back-end might be a database (as with IBM's SecureWay LDAP Directory), an X.500 directory, or something else entirely.

3.1. Directory Basics

The LDAP directory stores entries that are composed of attributes. Attributes have an associated type and some number of values.



An entry is identified by a Distinguished Name (DN). A DN is a sequence of attribute-value pairs called Relative DNs. Consider this example:

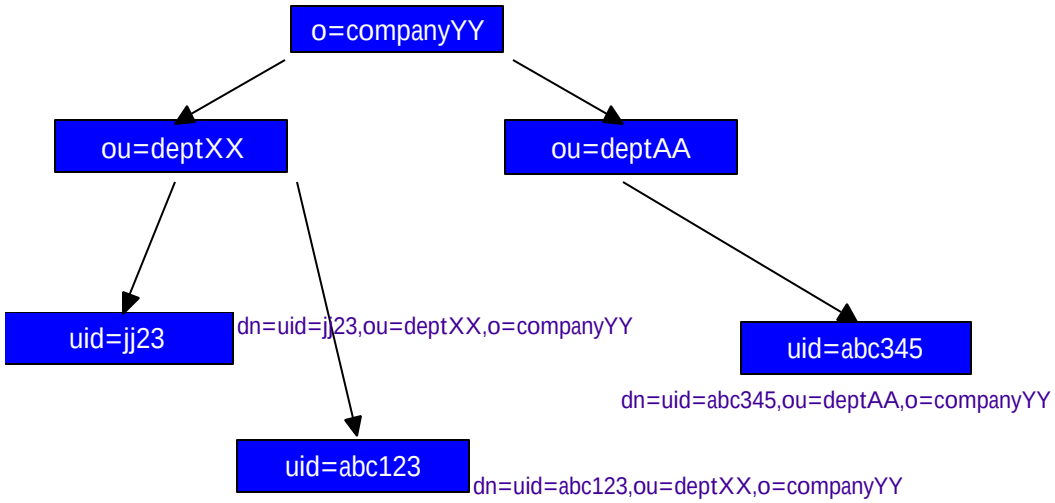
uid=kbotzum, o=ibm.com, c=us

The entire string is the DN for a particular entry (DNs are unique). The DN is composed of three RDNs which are themselves attribute-value pairs. In this case, there are three attribute types: uid, o, and c, which are user-id, organization id, and country, respectively. The entry itself most likely consists of some additional attributes. For instance:

cn=Keys Botzum
telephoneNumber=555-1212
userPass=secret
objectClass=ePerson
objectClass=inetOrgPerson

The first three entries are fairly obvious attributes (CN means Common Name) and values. The last two items specify the class of the entry. Thus, entries have types as well. Common types are defined as part of the LDAP standard and can be used by multiple vendors. A class defines the set of attributes associated with an entry.

Although it may not be obvious, the DN structure implies a hierarchical directory structure. For example, I might choose to create a structure for a department in a company with a base DN of the form ou=deptXX,o=companyYY (ou means organizational unit). In some sense, they are "under" that directory. Graphically:



3.2. Queries

An LDAP directory is searchable using a well-specified search string syntax . For details, look in the references given at the end of this document. Rather than an explanation of the syntax, here are a few handy search examples:

- `(&(uid=keys)(objectclass=inetOrgPerson))`– search for people with a userid of `keys`. Literally, this says search for all entries that are of type `inetOrgPerson` and contain the uid of `keys`. Under normal conditions, this should return exactly one or zero matches since a uid should be unique.
- `(&(cn=managers)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)))` – find all groups with the name `managers.` In LDAP, groups are normally represented as the object classes `groupOfNames` or `groupOfUniqueNames`.

3.3. Extensibility

Extensibility is not covered here, but you should know that the LDAP directory schema is extensible. Additional object classes and attributes can be defined. Arbitrary attributes can be attached to existing objects. The key point here is that LDAP and LDAP directories have the ability to store far more than just security information. What you choose to store will depend on your organization's needs. However, that extensibility also has a downside. It may be tempting to view your organization as somehow unique and attempt to define security attributes in LDAP that are unique to your organization. This is not a useful exercise in most cases. Commercially based products support and understand the standard LDAP security attributes. Adding arbitrary extensions is likely to reduce the value of commercial products and increase your own costs.

3.4. Authentication and Access Control

In addition to verifying user's passwords for the use by other systems (like a Web server), when a client binds to LDAP, the client is identifying itself to the directory. Most directories support multiple ways of authenticating to the directory. The most common way is using **ldap_bind** in conjunction with a username and password.

Usually, LDAP directories allow anonymous queries of most information. However, it is possible to control access to information in your LDAP directory by specifying ACLs on the directory tree. These ACLs restrict what clients can access. This makes it possible to store sensitive information in the LDAP directory. Then only authenticated clients with appropriate access rights can successfully query the data.

How access control is managed is vendor specific. You'll need to consult your vendor's documentation. IBM's SecureWay LDAP supports extensive access controls on individual entries or entire parts of the directory tree (ACLs are inherited).

In addition to basic access control, many LDAP accessible directories support automatic filtering of data based on the security rights of the requester. The directory can require that servers authenticate to it before returning sensitive information. Further, the directory can silently drop information that the caller does not have the right to know. For example, when asked what security groups the user is a member of, the directory returns only the groups that the caller has the right to know exist[2]. This feature is a key element of compartmentalized security. If this type of security is important to your organization, include it in your LDAP vendor evaluation criteria.

3.5. LDAP and Password Storage

For an LDAP directory to support password-based authentication using **ldap_bind**, the directory must store user passwords much like most operating system registries do today. This is not a problem with standard LDAP schemas as the schema includes an attribute known as userPass for precisely this reason. Each vendor provides various ways of protecting passwords. For example, IBM's SecureWay LDAP directory can store the password encrypted using a reversible or non-reversible algorithm[3]. Further, since passwords are considered sensitive, only highly privileged users can even see the password information. This builds on the ACL support discussed earlier.

3.6. Avoid Reflecting Transient Organizational Structure

When creating an LDAP directory structure, it is very tempting to create a structure that reflects your current organizational structure. This results in deep trees with directory nodes near the top reflecting current departments in your organization. You should avoid this and keep the trees shallow. Do not reflect organizational structure in the tree. Otherwise, the usual frequent corporate reorganizations will require changing users' DNs. This is not desirable since all existing systems that depend on those DNs, including certificates issued by an internal certificate authority, will no longer be valid.

3.7. DN Uniqueness

Remember that the DNs need to be unique. New directory administrators often use common names (CN) as part of the DNs. This is not a good choice as common names are not unique. You may find it best to put the uid in the DN and then ensure that uids are unique. For example, my DN might be "c=us,o=ibm,uid=kbotzum." My record would contain a CN of "Keys Botzum", but that is not part of the DN. This pattern also encourages shallow directory trees as recommended earlier.

4. Basics of Certificates and Certificate Authorities

4.1. Selected information about X.509 v3 Certificates

X.509 V3 certificates can contain a variety of information as specified in the relevant standards (see Reference section). The two fields of interest for our purposes are:

- *Subject* - the name of the entity being identified
- *Issuer*– the name of the CA that issued this certificate

Additionally, it is important to realize that certificate names (the subject and issuer) typically follow the X.500 naming standard. This is convenient as LDAP also follows this standard. Thus, there is a well-defined mapping from a certificate to an LDAP entry as long as the CA and LDAP infrastructures are designed to work together.

[2] IBM's SecureWay LDAP V3.1.1 directory exhibits this behavior when certain groups are marked more sensitive than others. A review of the documentation of several competitive products indicates that they also support this functionality.

[3] A non-reversible password encryption algorithm is like the Unix crypt algorithm. A password encrypted this way cannot be decrypted. Instead, user's passwords are verified by encrypting the input password and comparing it with the stored value. The advantage is that, password compromise is more difficult.

4.2. Quick overview of a Certificate creation, authentication, and management

In a PKI environment, a Certificate Authority (CA) issues clients certificates. A CA is a trusted set of services that use public-key encryption to issue and sign certificates. The basic parts and actors in this are:

- *Public-key* – public part of public and private key pair.
- *Private-key* – private part of public and private key pair.
- *Certificate* – a digitally signed piece of information that "proves" identity. A CA signs a user's public-key with additional information, creating a Certificate. Because a CA signs it, the CA is verifying that the Certificate is for the user claimed.
- *Certificate requester* – something that needs a certificate. Typically, this is a person or service that needs a way to prove an identity. This information will be placed into the Subject field of the certificate.
- *Certificate Authority (CA)* – a trusted system that issues certificates. This information will be placed into the Issuer field of the certificate.
- *Registration Authority (RA)* – typically part of a CA system. This piece accepts requests for certificates and validates the information. This information is then passed to the CA.
- *Certificate Revocation List (CRL)* – a list of certificates that are invalid. The CA periodically issues this list. The list may be stored in an LDAP accessible directory or stored somewhere else. Servers either use the LDAP directory, or obtain the CRL via other means.
- *Key Database* – this is a file used by a client or a server and contains information needed to validate and create PKI based connections. At a minimum, the server key database must contain the server's private-key and the server's certificate signed by some CA. If client certificate authentication is being used, the server key database must also contain the certificate from the CA that is used by clients. The client key database must contain the certificate for a trusted CA (the one that signed the server's certificate) and may also contain the client's private-key and certificate.
- *Smart-Card* – A smart-card is a device that stores and protects a user's private-key and certificate. In order to authenticate using public-key technology, the client system accesses the user's smart-card via a smart-card reader and obtains the certificate and possibly the private-key. Most smart-cards are tamper resistant and are protected by some access password. By using a physically separate device, the user's private-key is carefully protected. This is important, as private-key compromise is quite serious.

The public and private key pair creation is based on various algorithms that have resulted from cryptographic research. The basic point is that the public-key can be provided and published freely. This is in contrast to shared-secret based cryptography where there is only one key that cannot be compromised.

The private-key is held as a secret by the owner and cannot be revealed to anyone. To maximize security, it is desirable to store the private-key on a smart-card. Compromise of a private-key is a very serious matter. At a minimum, certificates based on that key must be revoked.

5. Overview of Enabling use of Certificates for Security

In order to use public-key based authentication there are some necessary items that must be configured in advance. The exact details are product specific, but the concepts are the same. Loosely, one must select a certificate authority (possibly one managed internally), configure the server with its own certificate for identification, configure the server with a CA's certificate, configure clients with the CA's certificate, and provision clients (or really individual users) with their own certificates. Much of this is beyond the scope of this document, but the steps below give a brief outline of what must be done from the server's perspective.

1. Determine what certificate authority you will use for issuing the server's certificate. You can choose from:
   1) A well-known public CA (such as Verisign). This option is typically used for public Web sites, but is also appropriate for organizations that do not wish to run a CA.
   2) An internal CA trusted by your users. This option is typically chosen for internal systems that will not be access by the public.
   3) A self-signed certificate. This option is only appropriate for prototyping.
2. Create a certificate database using your server's key management tool.

3. Using a tool specific to your server, obtain and then import the certificate for the trusted CA. If you are using a public well known CA, it is possible that this has already been done.
4. Generate your private and public key pair. With some CA's you cannot create your own private-key. In that case, the CA creates the private and public key pair and provides them to you via some secure mechanism.
5. Create a certificate request for your new server. You will most likely be prompted for some basic information: server's country (C), state, organization (O), organizational unit (OU), and common name (CN). If you are uncertain what to specify, contact your CA administrator for details. The naming conventions are often important. It is likely that a given CA can only issue certificates with certain values for c, o, and ou.
6. Send the certificate request to the CA. Often just a simple text file is emailed. It is also possible that steps 3-6 are done together by your tool.
7. Wait for the CA to respond with a certificate. Depending on the type of certificate requested and security concerns, you may be asked to provide additional proof of your identity. Remember that the CA signed certificate is the CA's voucher for your identity.
8. Using the server tool, import the certificate into your key database. This certificate is essentially your public-key and some additional information that has been digitally signed by the CA using its private-key.

At this point, the certificate database contains the CA certificate and the server's private-key and certificate. The server will use its own private-key to prove its identity to clients. It uses the CA's certificate to verify the identity of clients that are trying to prove their identity.

6. A More Detailed Look

We now return to the original generalized discussion from earlier in order to provide a more concrete mapping using the real world technologies just discussed. The following diagram shows steps 2 through 7 from earlier mapped to the two most common cases: certificate-based authentication of servers in conjunction with password or certificate authentication for clients. The diagram is conceptual rather than precisely accurate. Dashed arrows refer to events that occur once before normal operations begin. A detailed description follows the diagram.

Note: Step 1 is skipped, because it is not directly relevant to this discussion. In a typical system, SSL might be used to establish the secure tunnel. Web servers and WebSphere Application Server use SSL for this purpose.

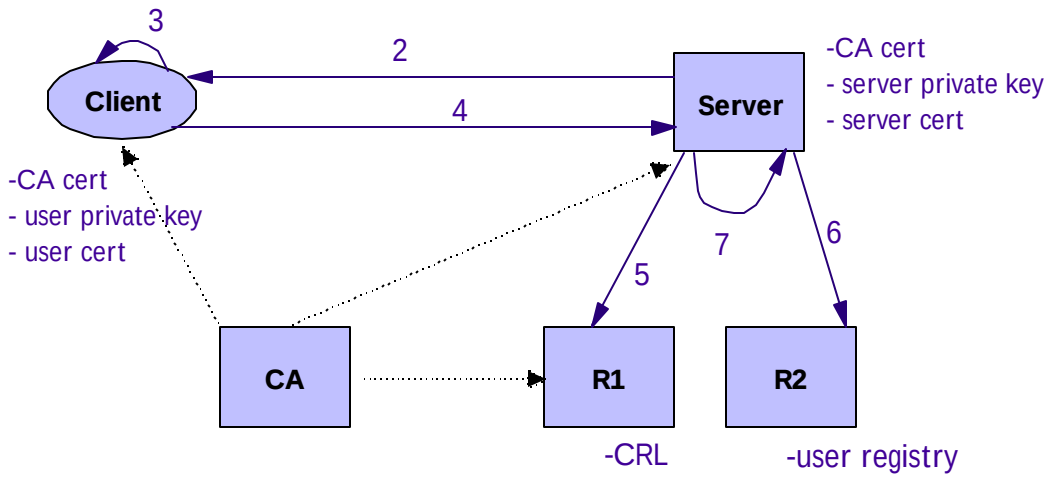


Diagram Discussion:

This section describes the diagram in more detail, focusing on password and certificate authentication.

Common to Both (recall that server authenticates via a certificate in both cases):

- In step 2, the server presents to the client its identification information. In a typical environment using SSL, this is the server's public-key certificate. This step actually occurs as part of the establishment of the SSL session, but it is logically separate.
- In step 3, the client verifies the server's identity by validating the server certificate and the server's proof of possession of the private-key that corresponds to certificate[4]. The certificate is validated using the CA certificate possessed by the client.

Certificate authentication for clients (also referred to as public-key authentication)

- In step 4, the client presents to the server its public-key certificate and proof of possession of the corresponding private-key.
- In step 5, the server verifies the user's certificate. First, it must ensure that the client really possesses the private-key. Having done this, the server ensures that the certificate is still valid. It can check a Certificate Revocation List (CRL). Logically, this list is stored externally in a registry. The original source of the CRL is the CA. In practice, the CRL may be stored locally and updated periodically, or it may be stored in an LDAP accessible directory.

Password Authentication (often referred to as basic-authentication)

- In step 4, the client presents to the server its user name and password
- In step 5, the server verifies the user's name and password against some registry (R1). This could be a local password file, but in an enterprise environment, the registry is maintained centrally and is thus external. The most common registry for this purpose is an LDAP accessible directory. The LDAP directory will validate the username and password by verifying it against information stored in the directory.

Common to Both:

- Now that the user's identity is known, the server searches the registry (R2) for authorization information in step 6. Typically, the server searches an LDAP accessible directory for a matching username if using password authentication, or a matching X.500 Distinguished Name (DN) taken from the client certificate. Typically, the registry will return a list of groups that specify the client's access privileges. It is worth noting that the registry could return other information that was more specialized if needed by the server.
- At this point, all of the basic authentication steps have been completed. Each user request (including the first) goes through step 6 to authorize the client's access based on its authorization information.
- Step 7 is typically performed internally by servers since they know best how to secure their domain. Thus, Web servers secure Web pages, Java Application servers secure servlets and EJBs, and databases secure tables, rows, and other database specific items. It is possible to externalize the authorization decision by using an external authorization service such as IBM's SecureWay Policy Director. However, this paper is attempting to focus on commonly available techniques. External authorization is still a new technique that is not widely used, although it shows great promise.

Comments:

- The two registries (R1 and R2) will typically be one directory, but they could theoretically be separate.
- The dashed lines from the CA are intended to indicate communication that occurs before the start of this scenario:
  - The CA provides to the client the CA certificate and the certificate of the client. This is typically done manually.
  - The CA provides to the server the CA certificate and the certificate of the server. This is typically done manually.
  - The CA provides to the registry the CRL. It is desirable that this be automated since the list changes. This area of technology is still evolving and is immature.
  - This aspect of public-key systems is one of the strengths of public-key technology. The CA does not have to be continuously available since it is not involved in the runtime authentication process.

It is worth noting that in the certificate case, the client must have had access to the private-key. The private-key must be stored securely and should only be accessible to the appropriate user. For the greatest security, this requires the

[4] Essentially, each party proves it possesses the private-key corresponding to its certificate by exchanging with the other information that is encrypted using public-key technology. Since the keys are asymmetric, communication is impossible unless the party being identified has its private-key. For example, in order for the server to prove its own identity, it sends to the client its certificate. The client verifies the certificate and then sends to the server a piece of information encrypted using the server's public-key. Only a party that possesses the server's private-key will be able to decrypt the information.

use of smart-cards. Because these are currently not very common, deployment of public-key based client authentication is not as easy as it may seem.

7. Proposed Operational Model

Now that we have discussed the basic details of the two most common forms of securing applications, it is time to propose an operational model that shows the infrastructure necessary to make this possible. As before, this model is intended to provide conceptual guidance and is not intended to imply a specific physical implementation.

The motivation for this proposal is the securing of enterprise applications consisting of a large number of applications used in the same organization or company that share a common infrastructure. Sharing infrastructure has the potential to improve interoperability of applications, reduce costs, simplify user management, and improve overall system security. Of course, like any infrastructure effort, some personnel must deploy and manage this central infrastructure.

An LDAP accessible directory is advocated for two simple reasons: it is sufficient for most needs, and it is widely supported by commercial vendors. LDAP access protocols and client APIs are well specified and supported by many vendors. Most importantly, the specification provides strong vendor interoperability for client access. For example, while working on this paper, the author used the IBM HTTP Server to secure Web pages using information provided by a X.500 directory supporting LDAP V2. The directory is not listed as being supported by IBM, but works without trouble since the LDAP standard is well supported and both products follow it. According to product literature IBM's WebSphere Application Server, Oracle, IBM HTTP Server, AIX, Compaq Unix, BEA WebLogic, Netscape, many Certificate Authority vendors (including IBM's First Secure), the PKIX standard, and others embrace LDAP. Of course, LDAP is not nirvana, but it is a widely supported solution to a common problem: central control and access to corporate security and user data.

Below is a diagram showing the proposed operational model. The model incorporates what is intended to be a PKI infrastructure. Many consider an LDAP directory a crucial part of a PKI. Additionally, the LDAP directory can provide services to non-PKI applications with little extra effort.

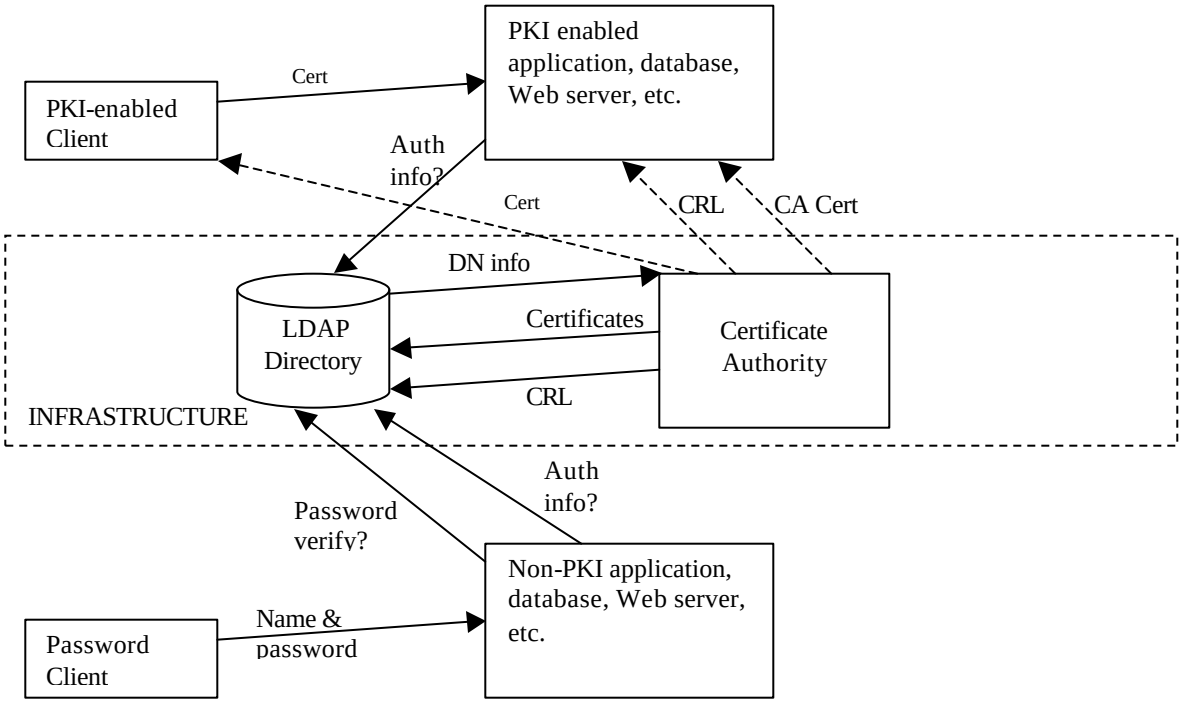


Diagram Discussion

- The dashed arrows from the CA to various systems are intended to indicate out-of-band communication. Information is sent to these servers via some non-real-time technique. In many cases, it is manual. The important point here is that the CA service does not participate directly in operational security checking.
- The diagram shows one LDAP directory. In practice a single logical directory is replicated to improve performance and reliability.
- The LDAP directory is more precisely an LDAP-accessible directory. The implementation is irrelevant. The LDAP standards are silent on directory internals. Thus, the LDAP interfaces can be supported by a X.500-based directory or some vendor-specific implementation. Many vendors use a relational database to store directory information since databases are well suited to this task and provide outstanding reliability and operational tools.

A key point of the diagram is that PKI enabled applications and non-PKI applications (those using passwords) are able to leverage the same infrastructure. For reasons that will be discussed later, PKI-enabling all applications does take time. By using an LDAP accessible directory as a key part of the PKI infrastructure, all applications can benefit from the information stored there. It is worth noting that password-based applications will benefit most from the LDAP directory if they are able to use the directory as part of the authentication process. Indeed, many products require this. To be useful for password-based authentication, the LDAP directory must store user passwords as discussed earlier.

Another benefit of the proposed model is that applications are benefiting from the centralized management of security information. Thus, each application team does not have to develop and maintain its own database of security information. More importantly, should a person's security rights change, only the central security directory needs to be updated, rather than many databases maintained by different application teams.

Some have expressed the concern that storing information centrally potentially increases risk since all information about a user is in one place. The concern is legitimate, but there are two alleviating factors. First, because the information is in one place, potentially expensive safeguards may be justified to protect this one resource rather than dispersing important information throughout the organization in an uncontrolled manner. Second, at a technical level, many LDAP accessible directories support automatic filtering of data based on the security rights of the requester. As discussed earlier, this makes it possible to limit directory access based on the security needs of the organization.

The most widely supported model of security is group-based access control. Virtually all commercial products that support access control use this technique. The standard LDAP schemas support storing group information directly using the groupOfNames attribute. Therefore, I strongly advocate using this model of security and, more importantly, driving security decisions from group information[5]. I expect that applications will continue to store rules locally until standards emerge to externalize rules as well. Of course, there may be some instances where the group-based model is not sufficient. If that is the case, LDAP directories can still be used for storage and retrieval since the schema is extensible. However, commercial products will be less likely to support non-standard techniques of securing data. You should be very cautious when choosing techniques that are less commonly supported and ensure that the need is truly fundamental.

Conveniently, the infrastructure proposed above is a widely supported model. Many products embrace it and can even provide some degree of built-in security control using such an infrastructure. A number of products, with no custom code development can secure access to resources (files, Web pages, servlets, EJBs, etc.) using either password-based or certificate-based authentication and then using an LDAP directory for authorization information. Enterprise application developers can then focus more on the business problem and less on writing endless lines of complex security code. Of course, not all products meet the exact needs of every application. If custom development is required, there are well defined APIs in Java (the JNDI API), C, and other languages for accessing an LDAP directory.

[5] If some more complex security technique is needed, such as multi-level access control, I still encourage the implementers to use information that is already available in a standard LDAP schema to drive these more complex rules. The reason is that complex access control applications and "normal" group-based access control can then benefit from the same centrally managed information.

8. Evaluation of Value

The model described in this document is my preferred approach to securing applications across the enterprise. The table below details some of the advantages and disadvantages of this and some other approaches. Definitions of terms used follow the table.

Advantages and Disadvantages	today	LDAP as part of PKI	Certs only	Certs w/ attribute certs	OS registry	DCE
Advantages						
Supported by many commercial products	X	X	X			
Based on open standards		X	X	X		X
Many commercial products can enforce authorization without any custom code		X				
Security information is managed centrally		X		X	X	X
Supports certificates	X	X	X	X		
Sensitive security information is protected by directory ACLs [6]		X				
Developers just do whatever is "easiest"	X					
Systems using password authentication still benefit from security infrastructure		X			X	X
Operating systems can use as registry [7]		X			X	X
Can easily update user security/authorization information centrally		X				X
Disadvantages						
Some group must manage central directory		X				X
Sensitive information is available centrally		X			X	X
User security/authorization information must be entered into multiple databases	X		X			
Each project team must manage own registry	X		X			
Users have to manage multiple certificates, or sensitive information may be inappropriately shared. [6]				X		
Operational systems depend on working central infrastructure [8]		X				X

Options Definitions:

- **Today** – The typical state of affairs today. Essentially, each project does whatever it deems reasonable. There are probably hundreds of different user databases in flat files, databases, operating systems, etc.

[6] Many LDAP directories are able to protect sensitive information by only providing it to requestors with a need to know. Since attribute certificates are simply passive data structures, if they are used for security information, users will either need one attribute certificate for every application, or sensitive information will be disclosed. For example, a user may present her attribute certificate to an application that only needs to know some of the information in the certificate.

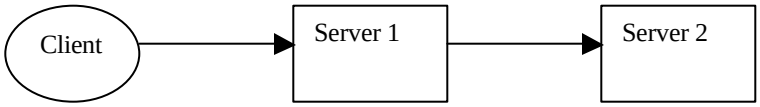
[7] AIX 4.3.3 supports LDAP. Compaq Unix is in the works and Windows 2000 uses an LDAP-based directory (Active Directory) for the registry. Of course, like many Microsoft efforts, they have added proprietary extensions.

[8] If a centralized directory is used and all applications require runtime access to it, the directory must always be accessible. Thus, the network to the directory must be stable and reliable and the directory itself should be managed as a highly available service.

- **Certificates** – This requires setting up a CA and Registration Authority (RA) to issue certificates for servers and users. To be truly secure, users should also be issued smart-cards containing their private-keys. Therefore all workstations must be equipped with compatible smart-card readers. There is no central directory of authorization information. Applications would only have the information available in the certificate. Typically, this is not sufficient, so applications would need an additional source of authorization information.
- **Certificates with attribute certificates** – Like the previous, but add the concept of attribute certificates. These certificates, also issued by a trusted CA contain the security information for each user. This information can be used to make authorization decisions.
- **LDAP as part of PKI** – Use a centrally managed LDAP accessible directory service for authorization and authentication information. The use of LDAP will allow authentication via certificates and passwords as appropriate. But, both techniques can still use the same authorization database (the LDAP accessible directory).
- **OS Registry** – some commercial products (WebSphere Application Server for example) can authenticate users against a local OS registry and use groups from this for authorization.
- **DCE** – use the DCE environment to build applications.

9. Security and Delegation

In a secure distributed environment, delegation is often an important topic. In a typical system, this type of interaction is common:



The client could be a Web server, a standalone client, or even some automated system. Server 1 knows the client's identity and then must contact Server 2 on behalf of the client. There are three ways of achieving this from a security perspective:

1. Server 2 simply trusts Server 1. Thus Server 1 can perform any action that any of its clients can perform. This approach increases application complexity by forcing Server 1 to do all security checking and, more importantly, increases the trust domain. If Server 1 is compromised, so is Server 2.
2. Server 2 simply trusts Server 1, but does allow Server 1 to specify the identity of the Client. It is important to realize that this is not truly more secure since Server 1 could lie to Server 2. However, this does simplify the development of Server 1 by allowing Server 2 to authorize access to its data based on the original caller, freeing Server 1 from this burden.
3. Client securely delegates to Server 1 credentials that it can use to talk to Server 2. In this case secure delegation is used, and Server 2 does not trust Server 1, but merely allows it to act on behalf of the Client. Appropriate security checks are done at each level based on the identity of the client. This is the most secure option.

Unfortunately, there are currently no popular standards for implementing secure delegation. DCE supported it, but DCE is not widely used. It is likely that a standard will emerge in the near future, possibly based on Kerberos. IBM WebSphere Application Server currently supports secure delegation from server instance to server instance via a proprietary mechanism known as Lightweight Third-Party Authentication (LTPA). WebSphere Application Server creates temporary secure tokens at a trusted security server that other WebSphere servers trust. WebSphere Application Server's token-based delegation is very similar to techniques used by DCE and Kerberos. Unfortunately, because the token is an IBM creation, the token cannot be used to securely delegate to a database or other third-party products. That would require support from the database and an open standard.

10. Why Support Password Authentication?

Another major question that comes up frequently is why supporting password-based authentication is an important criterion. The marketers tell us that PKI is supposed to solve all of our problems. The reason is simple: there are many gating factors slowing the move to an entirely PKI based authentication system. In addition to the usual political and policy issues, there are these technical and deployment factors:

- Smart-card reader deployment and support
- The roaming user problem
- Secure delegation
- Existing application upgrades.

The rest of this section discusses these issues.

10.1. Smart-Cards

In order to authenticate to a workstation using certificates, all workstations must be equipped with smart-card readers. Very few workstations have these readers today. Further, very few operating systems support certificate-based login. Passwords will be needed here for years.

In order to authenticate to a remote application using certificates, the application must be PKI enabled and the user's workstation must have access to the private-key. This requires that the private-key be stored on the local file system, or that the workstation supports a smart-card reader. If a smart-card reader is used, the client application must support the smart-card reader. If the private-key is stored on the local file system, it is subject to compromise since any user with access to that system can get to the key. Of course, the key is encrypted using a password, but relying on password encryption results in security more like password based security with the additional problem that private-key compromise is harder to correct.[9]

10.2. Roaming Users

Roaming users that choose to access computing systems from more than one location must have their private-key and certificate with them. Again, smart-cards solve this with the requirement that every workstation have a reader. If a smart-card is not available, roaming users must carry their private-key on a floppy disk or more frighteningly, store them in a network accessible way (distributed file system, FTP site, LDAP directory). I leave it to the reader to consider the security risks of a private confidential key being accessible via the network.

10.3. Delegation

To securely authenticate, the client in a PKI system must have access to the user's private-key. This can be problematic in the case of delegation. There are currently no widely supported ways of providing secure delegation, although standards are being developed.

Users sometimes have to access "remote" systems from their primary workstation. To securely access the remote system using PKI, the remote access tools (TELNET, FTP, etc.) must be PKI enabled. Updating tools will take time, assuming that it is possible. Moreover, after the user has accessed the target system, the user's private-key is no longer available on that system. Thus, secure applications that require PKI authentication cannot be started from that remote system. This last case may seem uncommon, but when a user starts an X-Windows session they are performing an analogous operation. Also, administrators often perform actions like this when working on several remote systems. This problem is closely related to the secure delegation problem.

10.4. Legacy

Lastly, few existing systems are PKI enabled. They will have to be modified or eliminated over time. Given the lifetime of software and technical barriers to PKI support, this will take a very long time.

Of course, all of these problems have technical solutions and will likely be solved. However, it is likely to be years before a perfectly seamless PKI environment can completely eliminate the use of passwords. Complete PKI based

[9] If a password is compromised, the user simply changes their password. If a private-key is compromised, the user must request a new certificate and private-key from the CA and the existing certificate must be revoked. All applications that trust that certificate must be checking for an updated CRL. Additionally, if the certificate's associated private-key was used for encrypting permanent data, the old private-key must be preserved or the existing data must be decrypted using the old private-key and then encrypted using the new private-key.

authentication systems are technically stronger than password based systems when implemented fully (including smart-cards) and should be pursued, but one must be realistic regarding the timeframes.

By supporting password based authentication against the same infrastructure used by the PKI environment, applications and systems that are not PKI enabled can still benefit from the substantial advantages of a centrally managed security directory. Many vendors are aggressively integrating their products with LDAP.

11. Policy-Based Security Tools

Policy-based tools typically provide a set of application APIs and a set of central servers that implement security policies based on some rules engine. Rather than each individual server making security decisions based on security information acquired centrally, the servers contact the policy servers and ask if a particular user is authorized to perform a particular action. This technique is advantageous for several reasons:

- Security policy tools often allow for complex security rules. For instance, it might be possible to allow certain users access to certain data only during business hours.
- For custom-built applications that must hand code authorization, a security policy tool frees the developers from having to develop this code. The server developers insert a small number of API calls to the central policy servers that do most of the work.
- The policy tools often include GUI tools that make it easier to specify and understand policies. In general, I advocate the use of such tools. IBM provides an excellent product for policy management: SecureWay Policy Director.

Today, the area of policy-based security is very immature and is not yet widely supported. This area even lacks widely supported standards-based APIs[10]. More importantly, many commercial products already have some level of built in authorization (Oracle, DB2, WebSphere Application Server, WebLogic, Netscape, Apache, etc). In order for those product to leverage a central policy management tool, the policy tool provider will either have to develop customized support plug-ins for each product, or the product vendor will need to add specific support for third-party policy tools. Due to the immaturity in this space, all products will not support any particular policy tool for some time.

Therefore, it is not appropriate for an entire organization to try to solve the security authorization problem by requiring the use of a single policy management tool. Rather, the organization must provide a complete central security infrastructure, consisting of certificate authorities and LDAP accessible directories. Then, additional policy-based authorization tools can be added that leverage this infrastructure. Applications that are able to take advantage of the policy tools can do so. Other applications can still leverage the core security infrastructure.

12. Conclusion

Application security is a large and complex topic. This document explains some of the basic facets of securing applications using modern techniques. More specifically, it shows the core role of an enterprise directory, an LDAP accessible directory. The use of an LDAP directory as part of a corporate PKI is desirable given its commercial viability and the problems that it solves. Within a so-called “pure” PKI environment, an LDAP directory is extremely valuable. In a realistic environment where password-based authentication must still be supported, an LDAP directory provides even more value. LDAP directories should be deployed as part of a corporate PKI. Such an environment will provide a more complete solution for securing realistic applications than a certificate authority alone.

[10] IBM submitted its SecureWay Policy Director APIs for fast track standardization by the Open Group and the API has been accepted.

APPENDIX

These appendices are intended to provide some more detailed technical information on securing enterprise systems as well as pointers to more information. They are not intended to be a complete tutorial on these issues, but will hopefully get you started.

Appendix A: Configuration Details for IBM HTTP Server (V1.3.6.x)

First, a key database for certificates must be created to enable SSL.

1. Ensure that you installed the IBM GSK toolkit. The toolkit is included in the HTTP distribution.
2. Run the IBM **ikeyman** tool. The executable may be named **ikeyman** or **ikmgui**.
3. Select Key Database=>File=>New to create a new database. Create a CMS key database. Choose an appropriate name and specify the password for the stash file. The HTTP server uses the password stored in the stash file to access the key database. This file will be placed in the same directory as the key database.
4. Import into the key database the certificate of the CA used by your users. In the content panel, select Signer Certificates and then click Add. If you are going to use a popular public CA, its certificate may already be in the database.
5. Now, create a certificate request by selecting Create=>New Certificate Request. Specify the field information as discussed earlier. The request is stored in a text file as ASCII. Send the file to your CA appropriately. Often, it is sent via e-mail.
6. In the content panel, select Personal Certificate Requests. You should see your request recorded there.
7. When you receive the certificate from your CA, add it to the key database. In the content pane, select Personal Certificates and then choose Receive. Specify the name of the response file here.

At this point, your server has its certificate and the certificate of a CA that provides certificates to clients. This process ensures bi-directional authentication when using certificate authentication. Clients that do not choose to authenticate via certificates still benefit from your server having one because certificates make SSL communication possible and ensure that clients can identify the server.

Now, using the administrative Web server, enable SSL and Certificate authentication:

1. Start the HTTP admin server if you haven't already done so. On NT, it is a service. On Unix, use the **<install>/bin/adminctl** command. Before starting it on either platform for the first time, make sure you specify an admin password using **<install>/bin/htpasswd** and put it in **<install>/conf/admin.passwd**.
2. Connect to the administration Web server using your Web browser.
3. Choose Basic Settings/Module Sequence. Then, select add to add the "ibm_ssl" module. Don't forget to click submit.
4. Choose Security=>Server Security and select Enable SSL. Specify the name of the key database you created earlier using **ikeyman**.
5. Choose Basic Settings=>Advanced Properties and add port 443 as an additional listening port for the Web server. The https protocol uses 443 by default.

SSL is now enabled. Clients will be able to determine the server's identity and client's can be challenged to provide their identity via a certificate or basic-authentication, but the server has no way of verifying their authorization.

Finally, enable LDAP:

1. Start the HTTP admin server if it is not already running.
2. Connect to the administration Web server using your Web browser. This is done by connecting to the target Web server host on a special administration port. Refer to the IBM HTTP documentation for details.
3. Load the module "ibm_ldap" by choosing Basic Settings=>Module Sequence.
4. Create a scope for authorization by selecting Configuration Structure=>Create Scope. Most likely, you'll create a directory scope. Make this the current scope by clicking on the scope button.
5. Create an **ldap.prop** file specifying LDAP info. Go to the **<install>/conf** directory and copy the sample **ldap.prop.sample** file to the name **ldap.prop**. Examine the file and edit appropriately. Most likely, you will

have to specify the URL for your LDAP directory. The example file at the end of this section configures LDAP security to support certificates and basic authentication.

6. Return to the admin Web interface. Set security in the current scope by selecting Access Permissions=>General Access. Select LDAP and specify the location of the **ldap.prop** file you just created. Specify an appropriate realm name that is meaningful to you. A browser user will be prompted with this name.
7. Restart the Web server. LDAP is now enabled.

Tips:

- You must hand edit the **<install>/conf/httpd.conf** file to specify the groups & users that can access directories. You will use the new **LDAPRequire** directives as shown below.
- You can choose to only enable LDAP. It does not require SSL, but then client communication (including passwords would be insecure).

Configuring LDAP Queries to use SSL for Secure Transport

To ensure that user passwords and other security information are not compromised on the network, the LDAP client can be configured to use SSL via the **ldap.prop** file. Here are the steps:

1. Ensure that your LDAP server is configured to accept SSL connections.
2. Create a key database using the **ikeyman** tool for use with the LDAP SSL connections. If your Web server is already configured to use SSL, you can use the same key database. The key database file must contain the certificate for the CA that signs the LDAP server's certificate. It can optionally contain the private-key and certificate for the Web server.
3. Place into a password stash file the password for the key database. Use the **ldapstash** program that is part of the HTTP Server to create the stash file (./ldapstash <password> <stash file>). Note: the stash file created by **ikeyman** will not work.
4. Edit the **ldap.prop** file as follows:
   - Change **ldap.transport** to 'SSL'
   - Change the existing **ldap.URL** to add a ':636' as the target port. LDAP SSL listens on a different port from standard LDAP.
   - Put the name of the key database on the **ldap.key.fileName** property and the stash file on the **ldap.key.file.password.stashFile**.
5. If the Web server is to authenticate to the LDAP server, update some additional properties. Otherwise, anonymous access will be used.
   - To use password-based authentication to the LDAP server, change the value of **ldap.application.authType** to 'basic' and update **ldap.application.DN** and **ldap.application.password** appropriately.
   - To support certificate-based authentication to the LDAP server, change the value of **ldap.application.authType** to 'cert' and add the name of the certificate that the LDAP client should use for authentication to the **ldap.key.label** property.

Here are excerpts from the two configuration files after enabling SSL and LDAP:

<install>/conf/ldap.prop file

```
ldap.URL=ldap://anldap.bigCorp.com/o=bigCorp.com
ldap.user.authType=BasicIfNoCert
ldap.user.name.filter=(&(objectclass=person)(uid=%v1))
ldap.user.cert.filter=(&(objectclass=person)(cn=%v1))
ldap.group.name.filter=(&(cn=%v1)(|(objectclass=groupofnames)(objectclass
    =groupofuniquenames)))
```

In general, you won't have to do much editing to the **ldap.prop** file. In the simplest case, you will just specify the URL of your ldap server and the **authType**. In this example, specifying "BasicIfNoCert" means try certificates first, and then fall back to basic/password authentication if certificates fail. Depending on your environment, you may also have to modify the search rules (as shown above). The lines above are the only lines I edited.

If you need to match on more complex parts of the certificate, look at the sample **ldap.prop** file. You can use the subject or issuer's common name, organizational unit, organization, country, and state. The subject's serial number can also be used.

<install>/conf/httpd.conf file

```
<Directory /usr/HTTPServer/htdocs/restricted>
AllowOverride  None
AuthName       "SecureWay LDAP Realm"
AuthType       Basic
Require        valid-user
#enforce security
LDAPRequire    group  apacheRestrictedAccess
LdapConfigFile "/usr/HTTPServer/conf/ldap.prop"
</Directory>
```

The above excerpt secures a single directory. In general, the directives are very similar to other HTTP security configuration except for the new **LdapConfigFile** directive and the use of **LDAPRequire** in place of **Require**.

Appendix B: Configuration Details for Netscape Web Server (V3.6)

Using a Web browser, connect to the Netscape administrative server (see Netscape documents for details).

To configure LDAP:

1. In the **Global Settings** section of the administrative interface specify that Netscape should use an LDAP directory. Specify the hostname of your LDAP directory and the base DN of the directory. Optionally, specify the use of SSL.
2. You can examine the Netscape directory configuration files. They are **<install>/admin-serv/config/dsgw-*.** The defaults may be sufficient.

To Configure SSL/Certificate Authentication:

1. Outside of Netscape, use the key pair generation program in **<install>/bin/admin/bin**. Run **sec-key** and follow the prompts to create a key pair file for later use. The alias you specified will be used again within Netscape.
2. Using the Web administrative interface, choose the **Keys and Certificates** section.
3. Select **Install Certificate** and import your chosen CA's certificate. To do this, specify that the certificate is for a "Trusted Certificate Authority," provide a name for the certificate and specify the file that contains the certificate. In the alias section, select an alias that you created earlier. Keep in mind, that if you will be using a well-known CA, there is a chance that the certificate will already be available.
4. Select **Request a Certificate** and send it to your CA. Use the alias from earlier. The information you fill in depends on your CA, so you should contact them. If the CA is accessible via e-mail, you can specify the e-mail address here. Make sure e-mail is configured to work correctly from your Web server. If it is not, the message will not be sent. In that case, save the request to a file and then send it to your CA manually.
5. When you get the (often via e-mail), put it in a file and import it into Netscape using the administrative tool by selecting **Install Certificate**. This time, choosing This Server.
6. At this point, your Web server has the certificate for the CA that your clients use and can thus verify their certificates, and it has its own certificate so clients can trust it.
7. Now, you must teach Netscape how to map certificates into the LDAP directory. Given a certificate, it needs to know what fields should be used to search the directory. To do this, edit the **<install>/userdb/certmap.conf** file. In the simplest case, you use one default search rule. Thus, you probably have to edit only the **DNComps** & **FilterComps** values. **DNComps** specifies the base DN for searching while **FilterComps** specifies the attributes to search with. The attributes are taken from the certificate. See the sample at the end of this section.

To secure a specific directory:

1. Using the administrative interface again, select Restrict Access=>Directory.
2. Select a directory you want to secure.
3. Add a new ACL and specify **use LDAP Registry**. If you want to use SSL, click the SSL box.

4. Watch the order on the ACL entries. Depending on your configuration, the order of allow and deny entries makes a difference. If things don't work how you expect, try moving the entries around (or learn what the ordering means).

Notes:
- You can examine the generated security files (which can be hand edited) in **<install>/httpacl**.
- For more details on the certificate mapping see the Netscape Server Administrative Guide.
- Unlike with IBM HTTP, I was not able to determine how to configure the Web server to fall back to basic authentication if the certificate is not accepted.

Here are excerpts from the userdb/certmap.conf configuration file after enabling SSL and LDAP:

```
#specify only one mapping, the default mapping
Certmap default default
#by leaving this blank, this says, "search entire tree"
default: DNComps
#search & match only on cn from certificate
default:FilterComps  cn
```

Appendix C: Configuration Details for IBM WebSphere Application Server Advanced Edition (V3.0.2 and V3.5)

To configure security with WebSphere Application Server, simply:
1. Install WebSphere Application Server following the normal installation documentation. Essentially, you will install WebSphere Application Server using the install GUI, configure an administrative database, and then start WebSphere Application Server for the first time.
2. First, verify that the install is okay. The best way to do this is to start the administrative GUI, start the default server, and then test some of the samples as described in the documentation.
3. To enable security, use the **Tasks** tab in the administration GUI. Then, select configure global security settings, and start the task (click on the green light). In V3.5, a **Task** menu has replaced the tasks tab.
4. In the first tab, select the check box for **enabling security**. Skip the second tab.
5. In the third tab (authentication mechanisms), select LTPA. LTPA is required for the use of LDAP.
6. In the fourth tab (User Registry), you should observe that LDAP is the current registry. Now,
    1. Specify for the **security server ID** and **password** a username and password of your choosing. You can think of this as being like "root" for WebSphere. This id must be in the LDAP registry. The id you specify here is the short name, not the fully qualified DN. For example, specify "webs", not "o=ibm.com,uid=webs". Also note that WebSphere Application Server stores this username and password in a property file. Since this identity has nothing to do with the operating system, I **strongly** recommend against using the operating system root id.
    2. Select the directory type. This is a hint to WebSphere Application Server indicating what query strings should be used. If your directory is not listed, or you have unusual schemas, choose custom and specify appropriate query strings. See the LDAP section for details.
    3. Specify the host (IP or DNS name) for the LDAP directory. Specify the port if it isn't using the standard LDAP port (port 389).
    4. Since most LDAP directories can hold more than one tree of data, you must specify the **Base Distinguished Name**. This is just the root of your tree. Your LDAP administrator should be able to provide this name. It is the base part of other distinguished names. In many cases, it is some form of the company or organization name. Often, the base distinguished name corresponds with the suffixes that the LDAP directory contains.
    5. If attributes that WebSphere Application Server needs to access are not publicly searchable, you may need to specify the **Bind DN** and **password**. This is how WebSphere Application Server authenticates to the directory.
    6. If the information being sent over the network is sensitive, select **use SSL**. If you are allowing password-based authentication, selecting SSL protects user passwords from disclosure.
    7. Click Finish.

7. Restart the entire WebSphere Application Server environment. Simply select **Restart** from the topology view on the WebSphere Application Server domain.
8. When the admin server restarts, reconnect using the admin GUI. When prompted for your **username** and **password** supply the security server identity you specified earlier. After connecting to WebSphere Application Server, you can give other users admin access by using the **Assign Permissions** task and editing the **AdminApplication** permissions.

SSL

There is no need to specifically configure SSL with WebSphere Application Server as it supports internal SSL connections natively. It is worth noting that SSL connections via the Web are actually the responsibility of the Web server. If you do choose to connect to the Web server using SSL (following the previous instructions), you must add an alias to the virtual hosts that WebSphere Application Server supports. In the Topology view, select the virtual host (probably **default_host**), click **Advanced**, and add aliases with port 443 (the SSL port). This will enable the servlet part of WebSphere Application Server to recognize requests on the new port.

When you configure security, there is a checkbox for **SSL from client to Web server**. Check it to make SSL mandatory and not optional.

Certificate Authentication
WebSphere Application Server 3.0.2.x does not support certificate-based client authentication. WebSphere Application Server 3.5 does. Refer to the WebSphere 3.5 documentation for configuration details.

Appendix D: Configuration Details for IBM SecureWay LDAP Directory (V3.1.1.1)

Most of the LDAP installation process is automated, but here is a rough outline of the key steps:
1. Install DB2, but do not bother creating a DB2 instance or database for LDAP.
2. Install one of the following Web servers: IBM HTTP, Apache, Netscape, or IIS.
3. Install the IBM SecureWay LDAP binaries using the OS specific installer.
4. Run the SecureWay Directory Configuration Tool. On NT, this can be selected from the menu. On Unix, run **/usr/ldap/bin/ldapxcfg** (this requires an X display). You will need to run the tool as a privileged user that can create new DB2 instances.
5. The configuration tool will ask you a series of questions. You should accept the default choice for all of them unless you see a problem. Make sure you remember the administrator password and DN.
6. Select **Configure**. At this point, the tool will take several minutes to create a DB2 instance, start the instance, and populate databases in it.
7. Start the HTTP server. The LDAP install created a new alias "/ldap" for LDAP administration.
8. Connect to the HTTP server using a Web browser, by connecting to the URL: http://<host>/ldap.
9. You will be prompted to login. Enter the DN and password you specified during the install.
10. Choose the **suffixes** option and select add suffixes. Here, add the root suffix for your naming tree. Each LDAP directory server can have more than one root (depending on what data it maintains). This suffix should be in appropriate syntax and unique. For example, "c=us, o=ibm" is one possibility.
11. Restart the LDAP server by selecting **Server**, and then **startup/shutdown**.
12. Now, you can use the Directory Management Tool (DMT). On NT, start it from the startup menu, on Unix, run **/usr/ldap/bin/dmt**. It is an X-Windows program.
13. Log in as the administrator. Select Server=>Rebind and specify the id you used earlier.
14. Immediately, create some object that maps to your suffix. For example, if you created the suffix suggested earlier, use the DMT tool to create the organization c=us,o=ibm. Select Entries=>Add entry. Then, specify the entry name as the **RDN** (there is no parent) and select **Organization** (assuming it is c=us,o=ibm).

At this point, you have completed the basics of configuring the LDAP directory. Now, you must create entries for users and groups. Needless to say, this is very specific to your business.

If you are attempting to develop an enterprise wide LDAP directory, I recommend that you obtain consulting expertise and read the LDAP book mentioned in the reference section.

Notes:

- Neither the LDAP server nor DB2 are configured by default to restart on boot. On NT, you can update the service to do this. On Unix, you will need to create a script that is run at boot. To start the LDAP daemon, simply execute **/usr/ldap/bin/slapd** as root. To start DB2, you should run the **/etc/rc.db2** script. That script starts all auto-start instances. Use **db2iset** DB2AUTOSTART=TRUE to define an instance as auto-start.
- The LDAP server outputs debug information to **/tmp/slapd.errors** and sometimes **/tmp/cli.error**.

To enable SSL security for LDAP clients:

1. Use the IBM **ikeyman** utility to create a key database. Follow the instructions in the configuring the IBM HTTP Server.
2. Using the Web administrative interface, select **Server=>SSL**. Then, click on **SSL On** and specify the file location of the key database you just created.
3. Restart the LDAP server. If there are problems check the logs carefully and verify the key database file name. The Web administrative tool and the LDAP server itself use different mechanisms to find the key database file. In some cases, the filename will turn out not to be valid.

Appendix E: Oracle Database (V8.1.6)

Oracle added support for LDAP in version 8.1.6. This support provides for schema less user authentication to the database using information obtained from a central LDAP directory. However, this support is only for Oracle's Network Directory and Microsoft's Active Directory. The LDAP protocols are well specified and the ability to extend an LDAP schema is widely supported, so it is unclear if this limitation is merely a temporary condition or an indication of some fundamental issue.

Notes:

- Oracle 8.2 will add support for Netscape's LDAP directory.
- Oracle Advanced Security Option is required for the use of an LDAP directory.
- Oracle has an additional component called "Secure Access" that provides for more elaborate security on database information than the core product. It provides for multi-level security. This product is not currently LDAP enabled.

Appendix F: Acquiring Group Information Using LDAP

This sample Java program uses JNDI to query an LDAP compliant directory for group information. Notice that it first converts the user's uid into a DN and then queries for **groupOfNames** objects containing that member.

```
import java.util.StringTokenizer;
import java.util.Properties;
import javax.naming.*;
import javax.naming.directory.*;


/*
  LDAP Java program to fetch groups by uid and output

  by Keys Botzum, IBM Transarc, January 2000

  This code works, but error handling is limited. If something
  unexpected happens, a cryptic exception will be raised by the Java
  runtime. You'll probably need a programmer to figure out what
  happened.

  Requirements:
  1) JNDI classes and an LDAP provider classes must be in
```

```
  your classpath. The LDAP provider classes are part of IBM SecureWay
  LDAP in <ROOT>/lib/ibmjndi.jar.  IBM's WebSphere Application Server
  includes an implementation of JNDI. It is in <ROOT>/lib/ujc.jar.

  So, for this to work for me, I did this:
  export
     CLASSPATH=$CLASSPATH:/usr/ldap/lib/ibmjndi.jar:/usr/WebSphere/AppSer
     ver/lib/ujc.jar.

  2) The file ldapgroups.properties must be in the current working
  directory. It assumes the file is right there.

*/

class ldapgroups {

    /*
      Fetch an existing entry from the LDAP directory. Search based on
     uid.
      */
     public static NamingEnumeration fetchGroups(Properties props, String
     uid)
     throws Exception {

     //get fields I need from the properties file
     String basedn = props.getProperty("basedn");

     DirContext ctx = new InitialDirContext(props);


     // 1) find complete DN for principle in question
     //Query ldap directory for dn.

     String dn;
     {
         SearchControls control = new SearchControls();
         control.setSearchScope(SearchControls.SUBTREE_SCOPE);
         String fetch[] = { "dn" };
         control.setReturningAttributes( fetch );

         //search for entry.
         //looking for entries w/ matching uid starting from basedn.
         String queryString =
              "(uid="
              + uid +
              ")";

        NamingEnumeration results =
              ctx.search(basedn, queryString, control);

        //get the matching result.
        //Having found relative dn, I'll need to add on basedn.
        if (results.hasMore()) {
              SearchResult sr = (SearchResult) results.nextElement();
              dn = sr.getName();

              //since dn is relative to basedn, must tack on basedn.
```

```
            dn += ",";
            dn += basedn;
        } else {
            throw new Exception("No matching uid found");
        }

        //if the uid is not unique we've got problems
        //(more than one element).
        if (results.hasMore()) {
            throw new Exception("uid is not unique");
        }
    }

    // 2) having found DN, find groups

    //control how search done & what is returned
    SearchControls control = new SearchControls();
    control.setSearchScope(SearchControls.SUBTREE_SCOPE);
    String fetch[] = { "cn" };
    control.setReturningAttributes( fetch );

    System.out.println("basedn = " + basedn);
    //search for entry.
    //looking for entries w/ matching dn starting from basedn.
    String queryString =
              "(member="
              + dn +
              ")";
    // should you choose to acquire only groupOfNames objects (as
    // opposed to any grouping object), add this to the query:
     //             ") (objectclass=groupOfNames))", control);

    System.out.println("queryString = " + queryString);
    NamingEnumeration results =
        ctx.search(basedn, queryString, control);

    return results;
    }

    /*
     */

    public static void main(String[] args) {

    try {
        String uid;

        try {
            uid = args[0];
        } catch (Exception e) {
            System.err.println("Usage: java ldapgroups <uid>");
            return;
        }

        java.io.FileInputStream stream =
            new java.io.FileInputStream("ldapgroups.properties");
        Properties props = new Properties();
```

```
            props.load(stream);

            NamingEnumeration results = fetchGroups(props, uid);

            System.out.print("Groups: ");
            if (results.hasMore()) {
                 System.out.print(
                       ((SearchResult) results.nextElement()).getName());
            } else {
                 System.out.print("NONE");
            }

            while (results.hasMore()) {
                 System.out.print(", ");
                 System.out.print(
                 ((SearchResult) results.nextElement()).getName());
            }

            System.out.println();

        } catch (Exception e) {
            System.out.println("Lookup failed: " + e.toString());
            e.printStackTrace();
        }
        }
    }
```

The associated properties file:

```
#These properties are used by the ldapgroup class

#new
java.naming.factory.initial=com.ibm.jndi.LDAPCtxFactory
java.naming.provider.url=ldap://localhost
java.naming.security.principal=cn=root
java.naming.security.credentials=root
basedn=o=myroot
```

References: Relevant RFCs, Drafts, Java Standards, References, etc.

PKIX and LDAP IETF RFCs and Drafts:

- IETF RFC 2459 - Internet X.509 Public-Key Infrastructure: Certificate and CRL Profile. Includes a discussion of the format and contents of an X.509V3 certificate.
- IETF RFC 2527 - Internet X.509 Public-Key Infrastructure: Certificate Policy and Certification Practices Framework
- IETF DRAFT draft-ietf-pkix-roadmap - Internet X.509 Public-Key Infrastructure: PKIX Roadmap
- IETF DRAFT draft-ietf-pkix-ldap-v3 - Internet X.509 Public-Key Infrastructure: Operational Protocols - LDAPv3
- IETF RFC 2254 - The String Representation of LDAP Search Filters
- IETF RFC 1778 - The String Representation of Standard Attribute Syntaxes
- IETF RFC 2253 - Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names
- IETF RFC 2252 - Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions
- IETF RCF 2251 - Lightweight Directory Access Protocol (v3)
- IETF RFC 2559 - Internet X.509 Public-Key Infrastructure: Operational Protocols - LDAPv2
- IETF RFC 2255 - The LDAP URL Format

IETF RFCs and Drafts can be found at http://www.ietf.org/.
The Common Data Security Architecture (CDSA) is documented at http://www.cdsasecurity.com/.

Java Standards

- J2EE – overall specification for "enterprise" Java environment
- JAAS – API for accessing security information
- JDK 1.2 java.security package – includes certificate and basic security APIs. JAAS builds on this.
- JNDI – preferred Java interface for directory access. LDAP modules are common.
- EJB – while this isn't security specific, the EJB specification explains how to secure Enterprise Java Beans.

Java information can be found at http://java.sun.com.

Relevant IBM Products

- IBM SecureWay LDAP Server supports LDAP V3, including SSL connections.
- IBM WebSphere Application Server is a Java Application Server supporting servlets, EJBs (Advanced Edition only), XML, etc. For security, it supports LDAP authentication/authorization with basic-authentication and certificates.
- IBM SecureWay Trust Authority is a Certificate Authority (CA) solution that includes a Registration Authority (RA), management tools, and hardware based cryptographic storage.
- IBM HTTP Server is a Web server that supports LDAP and SSL. It is based on Apache and adds LDAP, SSL, and a few other items.
- IBM Policy Director is a policy management tool that works with LDAP and a PKI infrastructure to provide centrally managed policy-based access control.
- IBM KeyWorks is a programming toolkit that provides an implementation of the industry standard CDSA APIs for managing PKI authentication and certificates.

IBM Software information can be found at http://www.software.ibm.com.

Books & Articles:

- Understanding and Deploying LDAP Directory Services, Howes, Smith, and Good, Macmillan Technical Publishing, ISBN 1-57870-070-1.
- IBM Redbook: Understanding LDAP, SG-244986, June 1998. A good high-level overview of LDAP with some technical details on configuration.
- IBM Redbook: LDAP Implementation Cookbook, SG-245110, June 1999. Another LDAP redbook. Focuses more on implementation details and covers SecureWay LDAP V3.1.

- IBM Redbook: Java 2 Network Security, SG-242109. Provides an excellent discussion of Java 2 security issues and some discussion of PKI issues.
- LDAP and JNDI: Together Forever, March JavaWorld article. http://www.javaworld.com/javaworld/jw-03-2000/jw-0324-ldap.html

IBM Redbooks can be found at http://www.redbooks.ibm.com and internally at http://w3.itso.ibm.com/redbooks/.