Evaluation of Distributed Authentication, Authorization and Directory Services

Gombás Gábor

programtervező matematikus szak, nappali tagozat
Témavezető: **Frohner Ákos**

ELTE TTK, 2001.

In the last few years, use of computers in all areas of life became common and wide spread. Nowadays even the smallest companies have at least a few PC-category computers. Larger enterprises and universities may have thousands of computers connected by some kind of network. Maintaining and administering such systems became a complex issue as the number of administration personnel is usually far smaller than the number of computers.

The most common requirement for computer networks is central administration: there is a small number of administrators who manage access control to all computers. It is often desired that workstations should be easily replacable: in case of the hardware failure, the failed unit should be replaced with a new one and the work should continue as soon as possible. The other common criterion is that every user or some groups of users should be able to use all or some groups of machines.

All the above problems require the existance of some database which holds certain management information about the valid users and their access permissions. Certain parts of that information should be made available only to their owner and/or the administrators, so some kind of authentication is also neccessary. Accessing other resources and services over the network usually also requires authentication. Since there are a wide variety of resources and services available, it is often desirable if all of them can use the same authentication mechanism.

In this thesis I'm focusing on comparing and evaluating currently available directory and authentication services. My main focus will be the view of a UNIX system administrator, but I also try to discuss integration and usability with other platforms and applications as well.

Contents

Chapter 1

Terms and Definitions

1.1 Directory services

Fax is a very common form of media used in transferring business documentation. If I want to fax a message to someone, I often have to phone the person first to obtain their fax number, before I can fax the message. British Telecom started to provide a fax directory service, in order to obviate the need for this 'lookup' phone call. However, it was reported in the press during September 1992, that the service was being withdrawn, due to the administrative overhead associated with keeping the directory up to date. This leads us to the conclusion that it would be much better to allow the administration of a global directory to be devolved to the participating parties, rather than trying to co-ordinate everything centrally.

Computers have similar requirements to people in respect of directory access. In order to make a connection, or send a message, or whatever the driving application requires, the software (and hardware) needs an address to process. Whilst it would be, and is, possible to provide the application directly with the address of the remote entity, in practice it is found to be better to provide a level of indirection. This is achieved by giving a name to the remote service, computer or application, and providing the local application with that name. The remote name to address *look up* is then provided via a table, or a database or a directory. This shields the local application from changes of address of the remote entity caused by such things as reconfiguration, replacement of hardware, or migration of a service between different nodes in a distributed system. The service which maps the name into an address, which is in essence identical to the white pages telephone directory service, has been given various titles such as name server, white pages etc.

Earlier and simpler versions of directory services was often called *name services* or *network information services*. I will use these names occassionally when describing services that use these expressions in their native documentation but I will refer to them as directory services when making general remarks.

Directory services hold all information in `entries`. Entries may be organized into *tables*, *maps* (maps are basically two coloumn tables with the first coloumn

being the key and the second being the value associated with the key) or a *tree*. Tables or maps may be organized to higher hierarchies usually called *domains*. The complete hierarchy is often called a *namespace* based on that entries usually identified by their names.

Both directory and authentication services can be either proprietary or standardized. In the world of the Internet the main standardization body is the *Internet Engineering Task Force* (*IETF*). There are several IETF working groups working on different problem areas. These working groups publicate their efforts in *Internet drafts*, which once accepted, can become official *Request For Comments* (*RFC*) documents and thus *Internet Standards*.

1.2 Authentication and authorization services

In the world of computers, a common situation is when one entity (called the *client*) requests some operation to be performed by some other entity (called the *server*). The server normally wants to restrict certain operations to be available to certain clients only. So there is a need for the client to somehow provide it's identity to the server. This process is called *authentication*.

Authentication can be either one-way or two-way. One-way authentication means that only one participant (usually the client) authenticates itself to the other. One-way authentication is used if one participant can be identified by other means (for example, a service running on a well specified port on a host with a known IP address). Other usage of one-way authentication is common in secure Web access, where only the server authenticates itself. The idea behind it is that the information the server provides (or at least the initial part of it) is public, but clients want to make sure that it really comes from the server they query.

Two-way or *mutual authentication* is required when neither parties can be identified implicitly but trusted communication need to be performed. In this case, both the client and server identifies itself to each other. This process usually assumes the presence of some previously known information (public key of the other party) or a trusted third-party service.

Entities who can be authenticated as are often called *principals*. A principal can be a person, a machine or a service. Principals are identified by their name; the format and the meaning of such names depend on the actual authentication service being used.

Once a principal is authenticated to a service, the server usually assigns a set of rights about what that principal is supposed to do. This process is called *authorization*. Authentication and authorization are often handled together and sometimes confused. Because authorization depends very much on the service being requested it is difficult to deal with it generally. Except the case when an authentication system was designed to work with a specific application or application area, authorization support is usually restricted to some very basic functionality and individual applications must implement their authorization policy on their own.

Authentication systems are usually built around various crypto algorithms. There are two classes of crypto algorithms we are interested in: *encryption algorithms* and *digest algorithms*. Encryption algorithms can be used to provide *confidentality*: that is, information is disclosed only to users authorized to use it (knowing how to decrypt it). There are *symmetric encryption* (often called *secret key* or *shared secret*) algorithms where the same key can be used both for encryption and decryption, and *asymmetric* (also called *public key*) algorithms where the key for encryption and decryption is different (they are mathematically related but computing one from the other is very hard).

Digest algorithms are used to compute a small footprint (called a *digest* or *hash*) that can be used to identify the document. Good digest algorithms have the feature that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given prespecified target message digest. This feature makes digest algorithms useful for *integrity* checking (if someone tampers with the stored ot transmitted data, its hash will change) and digital signatures (so it is enough to encrypt the digest only and not the whole amount data to be signed). More on cryptography can be found in [BS].

1.3 Example

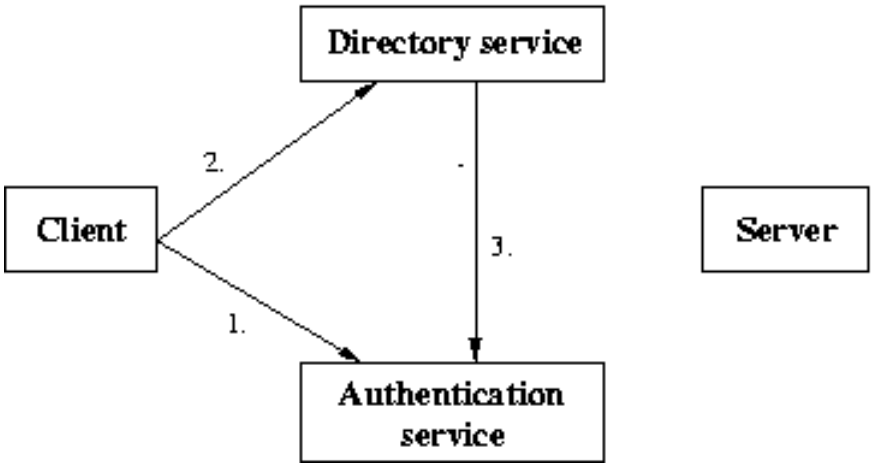

Figure 1.1: Location of a service

In this example a client wants to use a service offered by a server. Figure 1.1 shows that the client

1. Acquires the information needed to authenticate itself to the directory. This step may be optional since information about location of services is usually public.

2. Asks the directory for the physical address of the server providing the requested service.

3. When receiving the query, the directory verifies the client's identity using the authentication service. This step may be optional too just like the first.

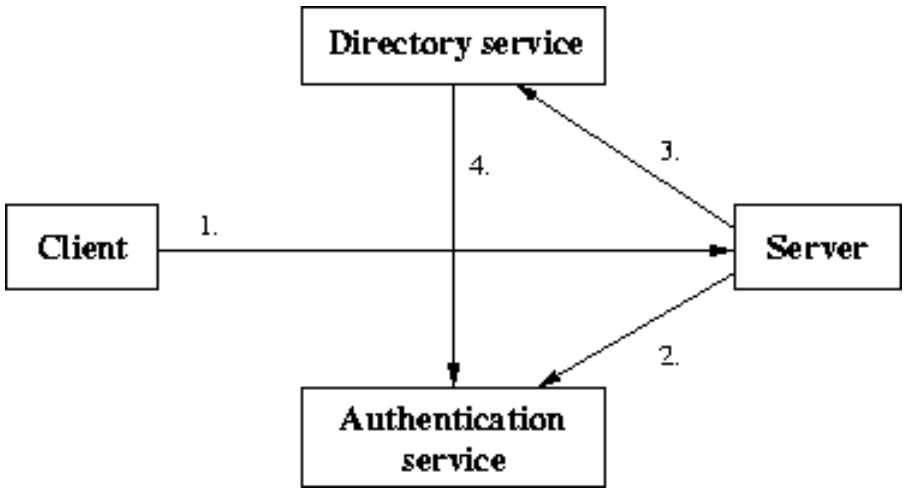

Figure 1.2: Usage of a service

The client then can send its query for the given service. What happens:

1. The client sends a request to the service.

2. The server checks the identity of the client with the help of the authentication service.

3. The server queries the directory for information needed to perform the request (this may include retrieving authorization information, locating other services or retrieving data only).

4. The directory checks the identity of the server (again using the authentication service) before returning the requested information.

This little example shows how an authentication and a directory service may interact with each other. Usually all of these interactions are transparent to the user; the actual methods used can be transparent even to the client and server processes if they are built using some generic abstraction layer.

Chapter 2

Tools

In this chapter I will give a short description of the authentication and directory services I wish to evaluate in the next chapter. The list is mixed: it contains API (Application Programming Interface) definitions, protocol definitions and actual products. I also included two abstraction layers (GSSAPI and SASL) which are not authentication services themselves, but they are widely used to interact with actual authentication services and therefore they have great importance.

2.1 Bootp, DHCP

The Bootstrap Protocol (*Bootp*) and the Dynamic Host Configuration Protocol (*DHCP*) are generally not referred to as directory services, but they have some properties that deserve our attention.

The original Bootp protocol was designed to allow a diskless client machine to discover its own IP address, the address of a server host, and the name of a file to be loaded into memory and executed ([RFC951]). Over time, several extensions was defined to allow other information to be embedded in Bootp responses such as the name of the client, IP addresses of gateways, name servers, time servers etc.

DHCP is designed to supply DHCP clients with the configuration parameters defined in the Host Requirements RFCs. After obtaining parameters via DHCP, a DHCP client should be able to exchange packets with any other host in the Internet ([RFC2131]). From the client's point of view, the DHCP protocol is an extension of Bootp. What is new compared to Bootp is the support for dynamically allocated IP addresses and a standardized extension mechanism.

[RFC2131] states that DHCP allows but does not require the configuration of client parameters not directly related to the IP protocol. This makes it possible to extend it to provide additional management information used for setting up a host during the boot process.

Bootp was widely used for supporting diskless X-terminals and small UNIX workstations not having enough disks for a local operating system install. DHCP become popular when the PC-world (dominated by Microsoft software) discovered

the Internet and wanted to be (or at least look like) compatible with existing IP-based administration tools.

2.2 NetBIOS, WINS

Before the Internet revolution, the *NetBIOS* service was the most popular mechanism for personal computer networking. NetBIOS is an interface specification rather than a protocol. Protocols supporting NetBIOS services have been constructed on diverse protocol and hardware foundations. Even when the same foundation is used, different implementations may not be able to interoperate unless they use a common protocol. This situation changed only when the support for the IP protocol family became more and more common and [RFC1001] finally defined a protocol standard over TCP and UDP.

NetBIOS was designed to be used by groups of PCs, sharing a broadcast medium. Both connection-orientated (session) and connectionless (datagram) services are provided, and broadcast and multicast are supported. Participants are identified by name, assignment of names is distributed and highly dynamic.

NetBIOS resources are referenced by name. Lower-level address information is not available to NetBIOS applications. An application, representing a resource, registers one or more names that it wishes to use. Management of the allocated names and location of named resources can be done by each client, or there can be a central NetBIOS name server (NBNS). The NBNS is often called as *WINS* which is the name of the most commonly used NBNS implementation made by Microsoft.

If there is no NBNS, clients generally use broadcast messages to communicate name registrations and locate resources. Even for relatively small networks this broadcast traffic may become a performance bottleneck. Using a NBNS this broadcast traffic can be reduced.

As said above, NetBIOS is used primarily in the PC world in a Microsoft environment. It is also generally limited to one subnet, though there are methods to access resources in other subnets. Due to this limitations NetBIOS/WINS is not a candidate for general directory services. However, it is used nearly everywhere where Windows-based computers are connected by some local network, so it cannot be overlooked.

2.3 DNS, Hesiod

In the 1970's, hosts connected to the ARPANET (predecessor of the Internet) used a single file called *HOSTS.TXT* that contained the name and address of all hosts. As the network grew it became apparent that manual maintenance of this file won't work in the long run and some other system was needed. This system was turned out to be what is called the *Domain Name System* (*DNS*) today.

The DNS is the most commonly used directory service nowadays. Nearly everybody who is using the Internet needs the DNS either explicitly or implicitly. It is defined as an Internet standard by the IETF in [RFC1034] and [RFC1035].

The design of the DNS makes it possible to store arbitrary data in it, not just name - IP address assignments. This capability was used by the Massachusetts Institute of Technology (MIT) for creating an information system called *Hesiod* using DNS as the database. Hesiod was part of the Athena project just like the Kerberos authentication service (see below).

Hesiod is essentially a YP-like service which uses the DNS to retrieve information about passwd entries, file systems, default printers, and so on - really arbitrary data stored as text strings. Queries and resource records use the HS class, and data are stored as TXT RRs. Hesiod is not designed to store security-sensitive data or support authentication; for these purposes, the Kerberos authentication service was created.

2.4 NIS

The *Network Information System* (*NIS*, formerly known as *Yellow Pages* or *YP*) was originally created by Sun Microsystems in the 1980's. It was designed to replace configuration data kept in files in the `/etc` directory (like `/etc/passwd`, `/etc/group`, `/etc/hosts`, `/etc/services` ...) on machines running UNIX operating systems. It uses the *RPC* (Remote Procedure Call, also by Sun) interface for communication between the client and the server.

The main problem with NIS is the lack of security. There is no authentication so every client can grab the whole database. In the case of password change, both the old and new password travel unencrypted over the network making it an easy target for sniffers.

The other thing that makes NIS unsuitable for large networks is that it has a flat namespace. If two or more NIS domains want to share information, it has to be duplicated.

NIS stores information in maps. Every map has a single key, so if some information has to be searched using different keys (e.g. the passwd map must be searched by name as well as by numeric uid), each key requires a separate map to be present.

2.5 NIS+

NIS+ is a network name service similar to NIS but with more features. Despite its name it is not an extension to NIS, it is a new product.

The NIS+ name service is designed to conform to the shape of the organization that installs it, wrapping itself around the bulges and corners of almost any network configuration. NIS+ enables you to store information about workstation addresses, security information, mail information, Ethernet interfaces, and network services

in central locations where all workstations on a network can have access to it. This configuration of network information is referred to as the NIS+ namespace.

The NIS+ namespace is hierarchical, and is similar in structure to the UNIX directory file system. The hierarchical structure allows an NIS+ namespace to be configured to conform to the logical hierarchy of an organization. The namespace's layout of information is unrelated to its physical arrangement. Thus, an NIS+ namespace can be divided into multiple domains that can be administered autonomously. Clients may have access to information in other domains in addition to their own if they have the appropriate permissions.

NIS+ uses a client-server model to store and have access to the information contained in an NIS+ namespace. Each domain is supported by a set of servers. The principal server is called the master server and the backup servers are called replicas. The network information is stored in 16 standard NIS+ tables in an internal NIS+ database. Both master and replica servers run NIS+ server software and both maintain copies of NIS+ tables. Changes made to the NIS+ data on the master server are incrementally propagated automatically to the replicas ([SOLNAM]).

2.6 LDAP

In the '80s an effort had begun to create a standardized directory service. The three largest organizations involved were the International Telecommunication Union - Telecommunication Standardisation Bureau (*ITU-T*, formerly known as *CCITT*) and the International Standards Organisation (*ISO*) and the European Computer Manufacturers Association (ECMA). The original problem the ITU-T faced was that e-mail users need to know the electronic mail address of other e-mail users. The other two organizations were concerned mainly with providing the name server service for Open Systems Interconnection (*OSI*) applications. The two tracks of development merged in 1986 by forming the Joint ISO/CCITT Working Group on Directories. The result was the International Standard ISO/IEC 9594-1 commonly known as ITU-T Recommendation [X.500].

As an aid to understanding what the information looks like, and how it is distributed and managed, various models are described in the Standard. Each model presents a simplified view of just one aspect of the Directory information. One model gives a view of the Directory information, as it is seen by a typical user. (This was, in fact, the only information model described in the '88 edition of the Standard.) This model, the Directory User Information Model, simply called the Directory Information Model in the '88 Standard, does not recognise that the Directory is distributed. From its perspective, there is a large amount of information held in the Directory, and users can access all of it, providing that they have the appropriate access rights. The Directory Operational and Administrative Information Model provides the 'administrator's view' of the information stored in the Directory. Administrators 'see' that there is more information stored in the Directory, than do typical users. This model still presents the Directory as a global infor-

mation base, and the distribution of the information between computer systems is not visible to the model. The Directory User Information Model, and the Directory Operational and Administrative Information Model, collectively make up the Directory Information Models.

Another model is concerned with how the information is distributed between the different computer systems that provide the Directory service. This is the DSA Information Model. The DSA Information Model describes a model of the information that needs to be held by a single computer system, in order for it to co-operate with others in providing a service to users of the Directory.

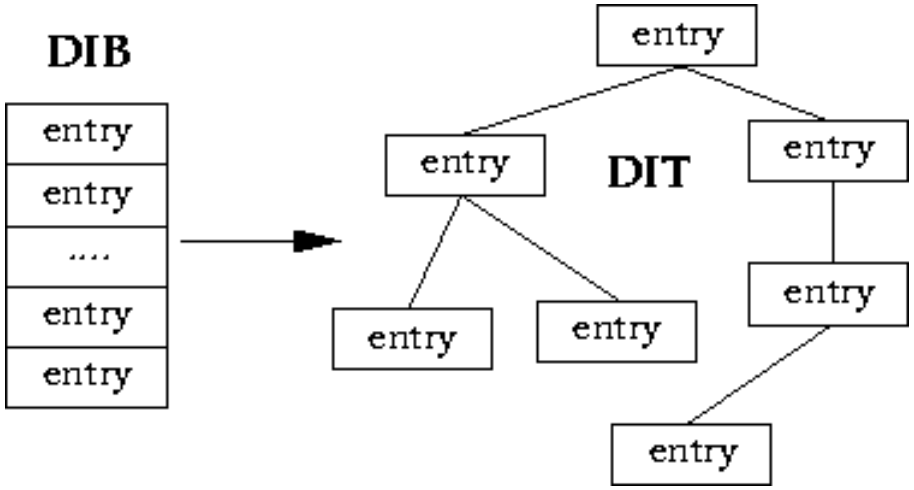

Figure 2.1: Stored data represented as a tree

The complete set of all information held in the Directory is known as the Directory Information Base (*DIB*). The DIB consists of entries, and these entries are related hierarchically to each other. In the Directory User Information Model, an entry holds information about an object of interest to users of the Directory. Entries held in the DIB are structured in a hierarchical manner, using a tree structure. The DIB can therefore be represented as a Directory Information Tree (*DIT*), in which each node in the tree represents a directory entry.

[X.500] had some big problems: first, there were virtually no commercial implementations. Second, the Directory Access Protocol (*DAP*) defined in the standard was defined over the ISO/OSI protocol framework while the world used TCP/IP. Third, the DAP protocol was rather complicated and required too much complexity in the clients to implement it. These problems led to the development of the Lightweight Directory Access Protocol (*LDAP*) in 1993 at the University of Michigan. First LDAP was used to access native X.500 directories, however LDAP proved to be the right tool for many problems outside the X.500 system. It was found that stand-alone LDAP daemons (those not relying on an X.500 back-end) could be used for a wide variety of directory based applications. Nowadays

the word "LDAP" is used not only for the protocol itself but for X.500-based directory implementations that use LDAP as their native protocol.

Due to this naming inconsistencies, I will use the word "LDAP" to refer to directory implementations providing a native LDAP interface and I will use the phrase "LDAP protocol" to refer to the protocol itself if it is not obvious from the context. I will use the OpenLDAP software for reference implementation.

2.7 ActiveDirectory

Together with Windows2000, Microft has introduced a new concept to the Windows world: the directory. Previous versions of Windows used different methods to store different data such as network or user information. ActiveDirectory was designed to unify all information about network objects in one common place. ActiveDirectory is based on X.500 and LDAP. It has an LDAP interface, although the schema used by Microsoft is not quite compatible with the LDAPv3 standard.

2.8 Novell Directory Service

The Novell Directory Service (*NDS*) serves the same purpose for Novell's Netware operating system as ActiveDirectory for Windows: it is designed to unify the storage and presentation of information of various network objects. It is also based on X.500. The NDS protocol is based on the DAP protocol just like LDAP; in fact, the LDAP interface provided by NDS is basically a simple mapping between the two protocols.

2.9 Kerberos

In 1983, at the Massachusetts Institute of Technology (MIT) work began on creating a distributed computing environment to allow users to seamlessly access services - it was called the Athena project. For this system to work, a convinient method of administering users and accessing services in distributed systems was needed. This need resulted in the creation of a distributed authentication, authorization and accounting service named *Kerberos* ([KRBPLAN]). The first public version called Kerberos IV was available in 1987. After becoming widely used, several security problems and design shortcomings shown up. The result was a redesign and redefinition called Kerberos V which appeared in 1993 [RFC1510]. The first complete reference implementation of Kerberos V was released by MIT in 1996.

The Kerberos authentication system is based on Needham and Schroeder's trusted third-party authentication protocol ([NS]) with modifications suggested by Dennis and Sacco ([DS]).

The authentication process proceeds as follows: A client sends a request to the authentication server (*AS*) requesting "credentials" for a given server. The AS responds with these credentials, encrypted in the client's key. The credentials consist of

1. a *ticket* for the server and
2. a temporary encryption key (often called a *session key*).

The client transmits the ticket (which contains the client's identity and a copy of the session key, all encrypted in the server's key) to the server. The session key (now shared by the client and server) is used to authenticate the client, and may optionally be used to authenticate the server. It may also be used to encrypt further communication between the two parties or to exchange a separate sub-session key to be used to encrypt further communication.

There are two methods a client can ask a Kerberos server for credentials. In the first approach, the client sends a cleartext request for a ticket for the desired server to the AS. The reply is sent encrypted in the client's secret key. Usually this request is for a ticket-granting ticket (*TGT*) which can later be used with the ticket-granting server (*TGS*). In the second method, the client sends a request to the TGS. The client sends the TGT to the TGS in the same manner as if it were contacting any other application server which requires Kerberos credentials. The reply is encrypted in the session key from the TGT.

Once obtained, credentials may be used to verify the identity of the principals in a transaction, to ensure the integrity of messages exchanged between them, or to preserve privacy of the messages ([RFC1510]).

2.10 Public Key Infrastructure

The ITU-T Recommendation X.509 (also known as Part 8 of ISO/IEC standard 9594) was developed as part of the directory standardization process mentioned in section 2.6. The original standard defined the form of authentication information that can be held in the X.500 directory, how such information can be obtained from the directory, and how to use this information to perform authentication tasks.

The X.509 standard defined two basic authentication methods: simple password based, and strong authentication. Strong authentication is based on Public Key Cryptosystems (PKCS). Public key cryptography requires the distribution of public keys of the participants. But this is not enough: given a public key, one has to know if it really belongs to the entity one wants to talk to - in other words, a public key and some identification has to be bound together. This task can be solved by introducing a trusted third-party *Certificate Authority* (*CA*). It is assumed that the CA has a public key known to all parties that wish to communicate securely. Therefore, the CA can issue a digitally signed statement called a *certificate* binding the participants' identity and public key together.

The *X.509* standard defines what information can go into a certificate, and describes how to write it down (the data format). The third revision (called X.509v3) was published in 1996 and is being widely used since then. This standard is the base of some major Public Key Infrastructure (*PKI*) systems used today.

There are several tasks a PKI has to be able to perform:

Authentication. Communicating parties need a way to verify the identity of each other.

Integrity. Digital signatures can provide integrity protection: that is, a digital signature is a proof that a document was produced by the entity having the secret key for a given certificate and it had not been tampered with during transport or storage.

Confidentality. Sensitive data has to be secured from unwanted monitoring. This can be achieved by using data encryption.

Non-repudiation . Especially needed in business applications where it is essential to have a method to ensure that somebody cannot deny a specific act she did in the past.

Certificate Authority. Certificates bounding public keys and the identity of their owners together has to be created and assigned.

Registration Authority. Before assigning a certificate the requestor's identity has to be verified. Certificates sometimes need to be revoked before their lifetime expires and such revocations must be tracked. The Registration Authority is often integrated with the Certificate Authority.

Certificate Repository. Having certificates is not enough. In order to communicate involved parties need to know each others' certificate holding their public keys. Therefore certificate repositories have to be set up where one can download the certificate of the entity she wishes to communicate with.

To perform these tasks, several algorithms, protocols, data formats and management recommendations has to be specified. In 1995, an IETF working group was formed to develop the neccessary Internet standards to support an X.509-based PKI. These standards are often named as PKIX standards after the name of the working group (which name in turn comes from the first letters of Public Key Infrastructure X.509). The PKIX standards include various portions of other standards such as the PKCS standard family created by RSA Laboratories.

Bseides PKIX there are other players on the scene as well. Since the comparison and analysis of different PKI systems could fill a thesis on its own I will focus on the PKIX infrastructure only but here I give some short notes about other systems.

SPKI/*SDSI* (Simple Public Key Infrastructure/Simple Distributed Security Infrastructure): This is a PKI which is not based on X.509. Now it is a joint effort

of the SPKI IETF working group and SDSI, an approach outlined by MIT's Ron Rivest and Microsoft's Butler Lampson. SDSI/SPKI differs from the more developed and accepted PKIX in specifying a highly distributed, client-focused trust model relying on delegated human-readable certificates. SDSI/SPKI also is more flexible than PKIX in letting end users define rules for processing certificates. It also rejects the complex ASN.1 syntax of X.509. Considerable control is put in the hands of end users, rather than relying on a centralized infrastructure for establishing identities. The infrastructure also puts an emphasis on short-lived, ephemeral certificates, reissued daily, for example, in lieu of extensive reliance on CRLs.

SESAME (Secure European System for Applications in a Multi-vendor Environment): This is an effort led by Bull, ICL and Siemens to create sophisticated single sign-on with added distributed access control features and cryptographic protection of interchanged data. SESAME is a construction kit. It is a set of security infrastructure components for product developers. SESAME uses the Kerberos protocol and some of the Kerberos data structures but also defines its own data structures. SESAME adds sophisticated access control features and the scalability of PKI systems to Kerberos. SESAME can be accessed through the GSSAPI with extensions to support the access control features.

Also there is a possibility to build a PKI around the famous PGP software. PGP has its own ideas about certificates and signatures and it is widely used for a long time. Using PGP as a PKI is often overlooked since it is not blessed by big standardization entities but uses much more informal definitions only. The PGP model has no need for central authorities but uses the "web of trust" method where everybody can decide who to trust and how much to trust. There are some short comings of PGP of course: since it was designed from the beginning to help secure messaging, it cannot be easily used for other purposes like strong authentication.

2.11 TACACS+

TACACS (*Terminal Access Controller Access System*) was designed by BBN to provide a centralized authentication, authorization and accounting facility. It has three major variants: TACACS, XTATACS and TACACS+. XTACACS is an extension of the original TACACS protocol, while TACACS+ is a new protocol developed by Cisco ([RFC1492]). The TACACS protocols are used mainly in networking devices such as routers, terminal servers and dial-in servers. From the above three variants TACACS+ has the most features so I will focus on it from now on.

In TACACS+, the authentication, authorization and accounting (AAA) services can be provided by the same or by different servers, using either the same or different databases for each. It is possible for example to exchange the authentication protocol with Kerberos and only use the authorization and accounting parts.

2.12 RADIUS

RADIUS (*Remote Authentication Dial In User Service*, [RFC2865]) was designed primarily for managing dispersed serial line and modem pools with a large number of users. RADIUS can authenticate a user and if it was successful, return configuration information specific to that user's session. The RADIUS protocol defines an easy way to add vendor extensions. This has several advantages because integrating new technologies and features is easy and the base protocol does not need to be revised. But it can also become a problem when different vendors start using the same extension for different purposes so interoperability can become difficult.

2.13 GSSAPI

The *GSSAPI* (*Generic Security Services Application Programming Interface*) is a generic API for doing client-server authentication. The motivation was that every security system had it's own API, and the effort involved adding different security systems to applications is extremely difficult with the variance between security APIs. However, with a common API, application vendors could write to the generic API and it could work with any number of security systems. Programs using the GSSAPI therefore can be highly portable, not only from one platform to another, but from one security setup to another and from one transport protocol to another. The GSSAPI provides several levels of data protection, consistent with the underlying security mechanism that have been implemented on a system ([SOLGSS]).

The operational paradigm in which GSSAPI operates is as follows. A typical GSSAPI caller is itself a communications protocol, calling on GSSAPI in order to protect its communications with authentication, integrity, and/or confidentality security services. A GSSAPI caller accepts tokens provided to it by its local GSS-API implementation and transfers the tokens to a peer on a remote system; that peer passes the received tokens to its local GSSAPI implementation for processing. The security services available through GSSAPI in this fashion are implementable (and have been implemented) over a range of underlying mechanisms based on secret-key and public-key cryptographic technologies.

The GSSAPI is defined in an abstract manner in [RFC2743]. Separate RFCs define the mapping of this abstract API to actual C and Java interface definitions and data types. The abstract definition makes it easy to implement the GSSAPI in other languages while the language-specific definitions ensure that different GSS-API implementations will be binary compatible with all applications.

2.14 SASL

SASL (*Simple Authentication and Security Layer*) is a generic protocol framework for doing various sorts of authentication between clients and server. In SASL ter-

mology, application protocols such as POP, IMAP, and SMTP specify a "SASL profile," which describes how to encapsulate SASL negotiation and SASL messages for that protocol. Different authentication schemes are called "mechanisms" in the SASL framework. The result is an abstraction layer between protocols and authentication mechanisms such that any SASL-compatible authentication mechanism can be used with any SASL-compatible protocol.

The standard SASL security mechanisms include the use of standard password based authentication (PLAIN), one-time passwords (OTP), GSSAPI/Kerberos, and X.509 PKI.

Chapter 3

Comparison

3.1 Objectives

There are several objectives to consider when evaluating and comparing different systems. The first thing is the purpose a specific system was designed for as this can be the major hint if the service can be used to solve a given problem or not. But new problems emerge every day and many of them can be solved using existing techniques. Understanding the existing solutions may provide guidelines for choosing existing products to solve problems that they were not designed for - simply because nobody thought about usability for that particular problem before.

The design goal is a very important but hardly measurable thing. It can help to narrow the list of possible solutions but there are a number of other objectives that has to be considered before making a decision:

1. The abstract mathematical model or models realized by a product. This includes the data model in the case of a directory service or the security model behind an authentication service. Obviously the problem one wants to solve must be possible to map to this model in order to have a chance for success.

2. Robustness and scalability. When planning a system, requirements about availability and error recovery have to be determined. The planned size and complexity of the system is also important. These parameters must be confronted first with the previously mentioned model's capabilities and then with the actual limits of the existing implementations.

3. Security. The times when the Internet was a place of cooperation have long gone and today's world requires the use of sophisticated techniques to protect properties and resources made available on the network. There is a wide range of security architectures. After the security requirements of a planned system are determined, one has to choose from the available solutions based on their applicability and interoperability with other parts of the system as well.

4. Supported platforms. In most cases people want to use existing solutions rather than inventing their own. The rationale behind this is that developing a new directory or authentication service is a very hard job requiring significant researcher and developer resources which few organizations or companies can afford. Using an already existing solution requires the availability of that product for the platforms the organization or company wishes to use.

5. Integration with other products. When designing the basic framework of directory and authentication services for a company or organization, it is important to investigate the possibilities of integrating the possible solution candidates with other applications and services in the future. Today the whole IT technology is changing rapidly and a more flexible solution might be useful if a new technology or service has to be supported in the future. Of course genericity usually is not free, its price has to be paid often by smaller performance or larger maintenance costs.

There may be other aspects when selecting between different solutions like policy decisions (all products must or may not come from a given set of vendors etc.). These are not covered here but in a given situation they may have as much or even greater significance as the objectives mentioned above.

3.2 Directory services

3.2.1 Design goals and basic principles

Generally it's a good advice not to use a software for a task it was not designed for. However, for general purpose products like some directory services, one may find interesting alternative usages with a little intuition.

Directory services can be either general or specialized. A general directory service like LDAP usually is more flexible and can be used for solving a much wider variety of problems than a specialized directory service, with the downside of usually being more resource-hungry. Specialization by itself does not mean that the directory service can not be used for other purposes as well - this is nicely demonstrated by Hesiod.

The Bootp and its successor the DHCP protocol was created to help the booting process of workstations connected to a network. The idea was to maintain configuration information required for starting the machine (such as where to load the operating system from) and basic networking parameters (such as IP address, netmask, addresses of gateways etc.). The original Bootp protocol was defined with mostly diskless X-terminals and workstations using UNIX-like operating systems in mind and was designed to work with a static configuration only. DHCP provides better support for other platforms like Windows and also has the capability to dynamically assign resources to newly attached hosts through a leasing mechanism.

The Domain Name System was designed to provide a distributed database for mapping host names to network addresses. It was needed because the previous method of having one big data file manually distributed was becoming unmanagable. The DNS offered a solution for dividing the network for smaller logical parts not neccessarily following the physical topology. These smaller parts were called *domains* and could have an administration on their own.

The DNS was designed to support a slowly changing database. Because changes are supposed to be infrequent, answers to previous queries can be cached locally for a long time which saves bandwidth and improves performance.

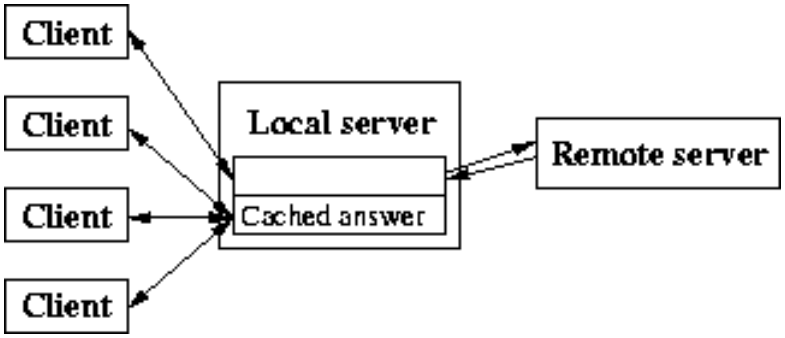

Figure 3.1: Caching the remote server's answer improves performance

Apart from name and address information, the DNS provides methods to store arbitrary data in it. This capability was used by MIT when they created Hesiod. Hesiod was meant to be a free alternative to other existing technologies such as Sun's Yellow Pages (also known as NIS). MIT wanted a system that could be freely distributed and could cope with the expected growth of the network in the next several years. They have choosen the DNS as a base because it had some very attractive features like the hierarchical namespace, the ability to delegate authority to subsidiary name servers and the ability to take advantage of local caching to improve performance.

Hesiod was designed to be a general database without knowledge about the stored information. It offers a content-addressible memory where certain strings can be mapped to others depending on the query. Hesiod has no knowledge about the data it stores, queries and responses are simple key/content interactions (as defined in [HESIOD]). Hesiod was designed for data that changes infrequently and must be accessed quickly. It can handle only small amounts of data per query so it is not suitable to be used as a general database. Hesiod has no interface to modify the data directly but relies on external maintenance and administration facilities.

NIS was created by Sun to replace the regular ascii files on UNIX machines that hold configuration information about users and services (like `/etc/passwd`, `/etc/services` and so on). The intent was to create an environment where these information can be kept and maintained on a central server and distributed to sev-

eral clients. The need for such a distributed system emerged shortly after the introduction of the Network File Service (*NFS*). NFS - like any other file system - uses numerical user identifiers and not names to identify users so if a filesystem has to be shared between different machines these machines has to agree on the user identifiers. The same goes for group identifiers too. Other than centralizing user management, NIS has the feature of storing data in hashed databases instead of flat text files which can speed up lookups if the case of a very large number of users. The tradeoff is the increased network traffic since NIS does not cache data.

NIS+ meant to be used for the same purposes as NIS but has several advantages over it. While the NIS namespace consists of independent flat domains, the NIS+ namespace is hierarchical. This allows splitting up a large network environment based on organizational boundaries. NIS supports one key per map only while NIS+ tables can have multiple keys thus eliminating data duplication. NIS+ offers advanced security features like authentication and access control which NIS does not have.

LDAP is based on the X.500 ([X.500]) directory standard. Originally it was meant to be a protocol that can be used in the TCP/IP world to access X.500 directories but it soon started to live on its own. Contrary to the previously mentioned directory services, LDAP meant to be generic without a specific predefined application area. The original intent behind X.500 was to provide a directory service for used mainly by humans to locate certain information. The original DAP protocol was complicated and hard to implement and required a full ISO/OSI protocol stack which is generally not available because the Internet is based on TCP/IP.

When native LDAP directory services have arrived it became possible to use it for low-level name services too and such applications soon emerged. For UNIX systems, [RFC2307] provides a schema recommendation which can be used to replace existing NIS and NIS+ implementations. There are both free and commercial products available that can automate the transition from NIS or NIS+ to LDAP (provided that you are not using proprietary solutions such as non-standard NIS+ tables).

Novell have choosen the same route on the creation of NDS as the creators of LDAP did. NDS is based on the X.500 standard just like LDAP and was aimed to provide a simpler and faster service than X.500. Novell wanted NDS to replace their previous information and management systems for their Netware operating system and later extended it to be able to store information about an entire enterprise network. When LDAP become popular, Novell voted to move into the direction of LDAP and made NDS compatible with it. This resulted in a change of NDS' goals: besides providing low-level name services to the operating system, more and more focus was given to high-level application integration in the e-business field.

When seeing how successful X.500 based directories became and understanding their advantages in centralized system management as well as in business applications, Microsoft realized that it must move in this direction too. The domain model that Windows NT previously used proved to be inadequate for large net-

works since it lacked very important features such as object hierarchy, extensible schema and data distribution strategy. Also, the proprietary solutions used for managing NT domains made it very hard to integrate it with non-Windows environments. The solution for these problems was the creation of ActiveDirectory, which uses DNS domains instead of the NT domains, provides X.500-based object naming and uses LDAP as it's communication protocol.

3.2.2 Data model

When evaluating a directory service, one of the most important aspects is the way it represents data. This includes what kind of data can it handle, how is it organized and how can it be retrieved.

There are three main data models used by directory services: *flat*, *hierarchical* and *tree*. The flat model (also called *flat namespace*) stores data in tables or maps which have no connection to each other. The data that can be stored in a table or map is usually key-value pairs requiring keys to be unique and the usual query methods being "get the value for key x" or "get all values". The set of tables that make up a separate database is often called a *domain*. Domains can have names and there may be more than one domain in a system, but these domains and their names has no connection with each other.

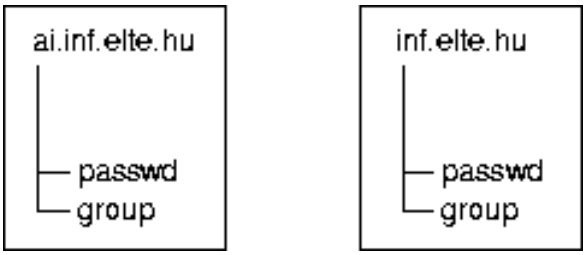

Figure 3.2: NIS: flat namespace

The *hierarchical* model (or *hierarchical namespace*) is an extension of the flat model. This model can be described as having several flat domains linked together usually in a tree. The classic analog for the hierarchical data model is the file system in UNIX-like operating systems: there are directories representing the domains and files representing the tables that contain the actual data. Domains can be named by their path to the root of the tree just like in the file system. Directory services using hierarchical data models usually make it possible to create some kind of relationship (either implicitly or explicitly) between different domains and/or tables inside these domains.

The third main data model is the *tree* model or *tree namespace* and it is the most generic. This model consists of objects linked in a tree structure. There is no predefined meaning of objects at specific parts of the tree, and each object

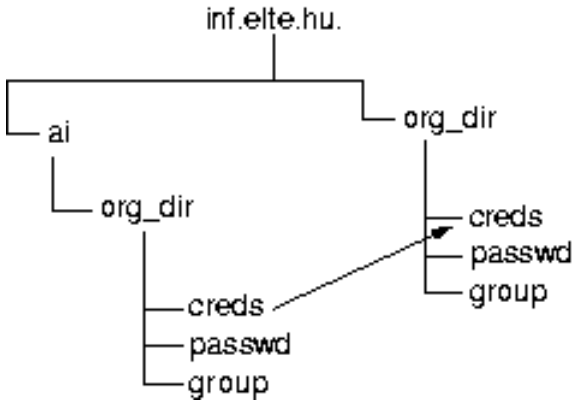

Figure 3.3: NIS+: hierarchical namespace

can contain information. This is unlike the hierarchical model where only tables residing in leaves of the tree structure contain information.

In the tree model, each object has its own (or relative) name. The whole name of the object is the path to the root of the directory, just like in the hierarchical model. Most directories support links or aliases to create shortcuts to avoid unnecessary data duplication. This can result in objects having more than one whole name and the graph of the data not actually being a tree but rather a directed graph without a directed circle (in theory, creating directed circles is also possible, but it is usually prohibited as it would serve no useful purposes).

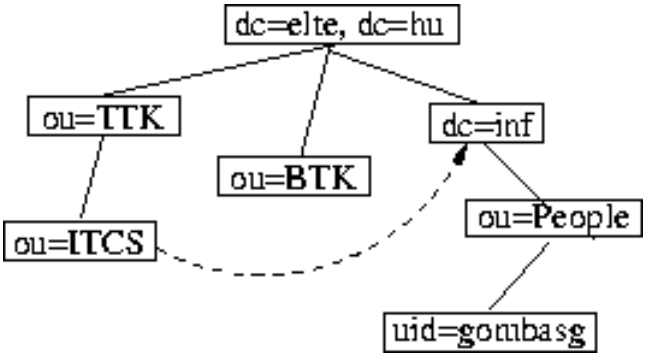

Figure 3.4: LDAP tree structure with a reference

Bootp/DHCP, NetBIOS/WINS and NIS all have flat namespaces. Bootp/DHCP and WINS do not even have the notion of domains or tables. They can do a single key-based lookup returning an information record associated with the key. While the Bootp/DHCP database is read-only, the WINS database can be dynamically changed as hosts register or deregister shared resources.

NIS has the notion of domains and maps. A domain consists of a set of maps and a map is basically a two-coloumn table, where the first coloumn is the searchable key and the second coloumn is the data associated with that key. So if some information needs to be accessed by more than one key then it must be stored in more than one maps. Duplicated maps are often named as the type of the stored information followed by the name of the key separated by a period (like `passwd.byname` or `passwd.byuid`). NIS tables are read-only.

NIS+ uses a hierarchical namespace which gives it several advantages over NIS - especially in large and complex network environments. Contrary to NIS, tables in NIS+ may contain more than two coloumns and any number of coloumns can be searchable. The full key of a row is the concatenation of the values in all searchable coloumns, and this must be unique - one searchable coloumn can contain the same value multiple times (provided that there are at least two searchable coloumns).

Tables in NIS+ has a special feature called concatenation path. It is used by search operations: if a key is not found in the actual table, tables listed in the concatenation path will also be searched. This makes it possible in large network environments that common information (like normal users) is only stored at the top level of the hierarchy and lower level domains contain information only that is specific to that domain. From the administrative viewpoint this is a nifty feature that can make system and network administration quite convinient.

DNS is often referred as having a hierarchical structure but by our above definitions DNS falls into the tree model category because not only the leaf nodes (in this case, host names) can have information (*resource records*) associated with them, but also the internal nodes (domain names). The tree is partitioned into *zones*; each zone starts at a domain boundary and extends downwards to leaf nodes or other zone boundaries. Resource records have a type parameter and a value. When doing a query, the client can specify what resource types is it interested in about a given name. There is a possibility for querying all entries of a zone but this capability is often administratively restricted to master-slave server communication and is not available to ordinary clients.

The DNS has support for pointers (`PTR` records) that can link different parts of the tree and aliases (`CNAME` records) which provide alternate names for nodes in the tree. `CNAME` records are meant to be used in general while `PTR` records are meant to be used with special domains only like the `IN-ADDR.ARPA.` domain used for reverse lookups.

The original assumption about the DNS was that it is a mostly static database with changes made externally by editing configuration files. This assumption became problematic when DHCP become widely used. If the machine of Bob got a dynamic IP address and he wanted to tell Alice to use some service from his machine, he could not tell her the name of his machine because the name servers had no knowledge about what IP address does he got. For these reasons a protocol extension for dynamic updates was defined in [RFC2136]. This extension allows atomic updates bound to specific prerequisities.

[X.500] defines several models of the directory including information, opera-

tional and administrative models. These models are also used by other directory services based on X.500 like LDAP, ActiveDirectory or NDS.

In X.500, the main concept of the information model is the *entry* (the corresponding term in the case of the DNS is the resource record). Entries are named collections of attributes. Attributes have a type and one or mode values, with an associated syntax defining allowed information. The range of required and allowed attributes are defined by a special attribute called *objectClass*. Each entry has exactly one structural object class but may have zero or more auxiliary object classes. The union of required and allowed attributes by each object class make up the set of requred and allowed attributes of the entry.

The [X.500] naming model describes how objects are represented in the directory. It is a tree model as defined in the beginning of this section. Each object has a *Relative Distinguished Name* (*RDN*) which is one or more (usually one) selected attribute/value pair of that entry. The full *Distinguished Name* (*DN*) of an object is made up by the RDNs of the objects in the path to the root of the directory tree.

One big advantage of X.500-based directory services over DNS is the modifiable schema. The DNS only allows a predefined set of resource records. Some of these records can hold arbitrary information but it is left totally to the client how to interpret it. In X.500 the structure of information about an entry can be expressed by using attributes and object classes.

The other advantage of X.500 over DNS is the wide range of supported operations. These include powerful search and comparison capabilities as well as add, delete and modify operations. The search functions can be used to locate entries satisfying a given search filter criterion in a subtree of the directory. This subtree can span across multiple servers which makes it easy to repartition a directory if the amount of data grows such that one server can not handle it.

All X.500-based directory services have support for splitting the directory tree they serve to smaller parts and having different servers serve different parts. Such a subtree is often called a *partition* (the equivalent of a DNS zone). Partition boundaries are often indicate organizational boundaries.

3.2.3 Robustness, availability, scalability

In most cases directory services are critical components of the network environment. If they fail, computers will not be able to boot, users cannot log in, e-mails do not get delivered and so on. When evaluating the robustness of a given directory service, several aspects has to be considered:

Replication. There are companies guaranteeing 99.9% availability for both of their hardware and software, but these solutions are rather expensive and even these solutions can fail eventually. So it is essential that a core service like a directory should not run on one machine only but have one or more replicas. If one server fails, the others can take over and provide continuous service without clients even noticing the failure.

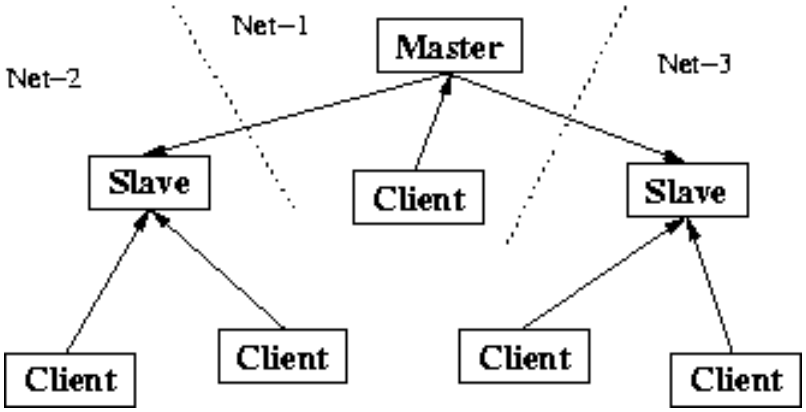

Figure 3.5: Replication improves scalability and reliability

Replicas can be either *read-only* or *read-write*. In a read-only replica setup, replicas handle data queries only, and one master server handles all the updates. It has the advantage that it can be easily implemented but it has the obvious disadvantage of placing all responsibility in making updates on a single server. This provides a *single point of failure* which can be unacceptable in some situations. Performace issues (the load for all updates is placed on one server, see below) can also arise.

Using read-write replicas has the advantage of eliminating the single point of failure and also solves the performance problems. The disadvantage is that read-write replication is hard to implement right. If clients are allowed to access more than one servers at the same time and they make modifications on more than one servers at the same time, the modifications may conflict and resolving such conflicts can be challenging.

Failover. Having replicas is not enough. If one server fails, clients should be able to automatically switch to another server. If it is not the case and administrative intervention is required to replace the faulty server, it can cause noticable downtime which can be measured in lost money if the service is business-critical.

Availability. As already mentioned earlier, directory services can be an essential part of a network setup. This means that the service must have a high availability ratio. Replication was already mentioned, but it is also important that the directory should not go offline for reconfiguration or backups.

Scalability. The amount of data a directory service has to handle can be very big and tends to grow over time. It may be impossible or inefficient to keep all data on one server only because the hardware required to handle the amount of data and/or the amount of client requests needed may be too expensive.

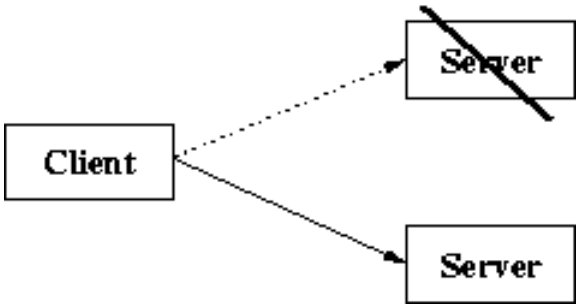

Figure 3.6: Failover on server failure

Latency can also be a problem: if the directory needs to be used over slow network connections the response times may be bad even if the server itself is fast. These problems can be resolved by putting a server near the clients (according to network topology) thus eliminating the need to use the slow connection.

Service	replication	failover	database size	clients
Bootp/DHCP	no	yes	100	100
NetBIOS/WINS	no	yes	100	100
DNS/Hesiod	r/o	yes	10,000	1000
NIS	r/o	yes	1000	100
NIS+	r/o	yes	10,000	1000
LDAP	r/o	yes	10,000,000	1000
ActiveDirectory	r/w	yes	10,000,000	1000
NDS	r/w	yes	100,000,000	10,000

Table 3.1: Replication capabilities and estimated scalability limits

Table 3.1 summarizes the capabilities of directory services. The database size means the number of entries a single server can handle on the proper hardware. The clients coloumn contains the number of clients the service is designed to handle easily. Both numbers are meant as a magnitude estimation only since they can vary (even significantly) between implementations and installations.

As to be seen, every directory service has failover capabilities. In the case of Bootp/DHCP it is achieved by the client sending a broadcast request and using the server that happened to answer first. NetBIOS clients can also operate in broadcast mode when the answer coming from the fastest server will be used, or it can be configured to try to use a specified set of servers in the order they are listed. The main difference between Bootp/DHCP and WINS is that while the former is normally used for boot-time configuration, the latter is used every time a network

resource needs to be located. So while broadcasting is fine for the Bootp/DHCP protocol, in the case of NetBIOS broadcast traffic can cause noticable performance problems even in relatively small networks.

Bootp/DHCP and WINS do not have replication capabilities. The former has a static database defined by a configuration file which must be transferred to other servers manually, the latter is a dynamic service where clients register themselves if they want to be found by other entities.

One of the big enchanchments of DHCP over Bootp is the capability of dynamic resource allocation. Bootp was designed to have a static configuration database defined on the server while DHCP can automatically assign resources such as IP addresses from a list as new hosts are attached. This makes DHCP capable to handle bigger network environments where adding or removing hosts is common so maintaining a static database would be hard. DHCP can also be very useful if the usable IP address range for some reason is smaller than the number of possible hosts but these hosts are not used in the same time. By using small lease periods a newly attached host can reuse the IP address of a host that was detached from the network earlier.

The DNS is one of the most heavily used distributed databases. It is the base of nearly all human-controlled internet services because people do not like to remember numeric addresses but want to use meaningful names wherever possible. It has a classic read-only master-slave replication model with a static (not modified by clients) database (although there is a dynamic DNS update protocol but not many vendors are using it). Because of these characteristics DNS seemed to be a good basis for a more general directory service at the time people at MIT designed Hesiod. There were some shortages: due to the DNS protocol, a very restricted limit (512 bytes) was imposed on the data the server can return for a given query.

In the case of NIS, there are one master server for each NIS domain, and there can be any number of slaves. When performing queries, the server that answers first to an initial broadcast request will be used. All changes happen on the master server. The basics for NIS+ are similar, but the preference of the NIS+ servers can be configured centrally. Servers with the same preference will be used in random order. Talking about scalability, NIS can handle a couple hundred of users easily but there are existing setups with several thousand entries. NIS+ is officially claimed to efficiently handle about 10,000 entries in a domain having no more than 10 replicas and about 1000 clients.

One of the big drawbacks of NIS is that the database can not be updated directly. The original sources of the maps are kept on the master server in text format and modifications are made to those files. The master server regularly sends out the new version of the maps to all replicas. This causes considerable network and performance drawbacks since the master sends the whole database to each slave every time, there is no method for incremental updates. It also means that updates have noticable and sometimes inconvinient delays.

NIS+ addresses the shortcomings of NIS mentioned above: there are direct ways to update the database eliminating the need to constantly convert from a

source format. Updates are incremental and are sent to replicas almost immediately so problems resulting from replica inconsistencies do not cause so much trouble. Incremental updates also result in lower bandwidth and CPU requirements.

The LDAP protocol itself does not define failover or replication. LDAP clients generally accept a list of servers to use for queries and will try the servers in the order they are listed until one responds. Another method for achieving failover is to use DNS tricks like multiple IP addresses bound to the same name; applications are expected to try the given addresses in the order they got them from the name server until one succeeds. The problem with the latter approach is that it is a notorius bug that many applications can not handle properly the case of multiple addresses per one host.

A standard replication protocol for LDAP (called LDUP) is being currently worked on but has not been finalized yet. OpenLDAP provides a simple replication scheme: changes in each partition can be sent to a given list of servers. It can not restrict replication to subtrees of a partition nor can it filter the list of attributes being sent to a replica. This is clearly a field when OpenLDAP falls behind commercial directory services. Currently OpenLDAP officially supports read-only replication only but the development version has support for read-write replication (called *multimaster* mode) as well. The importance of multimaster support increases significantly if somebody wants to build a high-availability environment. Read-only replicas can solve the case of information availability when a server fails, but they do not provide any help when a client wants to perform an update on the directory but the master server is not available.

With NDS, replication can be configured per partition. The NDS terminology calls an instance of a partition a *replica*. Each server can hold one or more replica, but every instance of a partition must be on a different server. Each partition has a master replica (by default on the server that created the partition) and can have several read-write and read-only replicas. Read-write replicas can help to make updates faster if there are slow WAN connections in the network, but they also generate more traffic due to the required synchronization. Read-only replicas can be used to speed up queries.

NDS only guarantees loose consistency meaning that replicas are not guaranteed to hold the most recent changes to the directory at any specific moment, but the database is guaranteed to synchronize eventually. Changes get propagated to other replicas using predetermined paths. This is possible because every replica has definite knowledge of all other replicas so it can calculate the full topology and can send changes to neighbour replicas only.

ActiveDirectory also has the notation of partitions just like NDS or OpenLDAP. Replicas can be either full read-write or partial read-only. Partial replication means that the replica does not contain all attributes of replicated objects but only a filtered subset. Apart from some minor differences NDS and ActiveDirectory has quite similar replication capabilities.

OpenLDAP, NDS and ActiveDirectory all share the feature that only the actual changes get replicated, not whole objects. This has obvious performance bene-

fits and also reduces the chance and scope of conflicts in the case when a client performs modifications on more than one read-write replica.

3.2.4 Supported platforms

There are two main platform groups that I consider: UNIX-like systems and Windows. Other operating systems are not so widespread and their usage is usually limited to specific tasks that do not require integration with generic directory services. Of course there are directory implementations for VMS and IBM mainframes, but I do not want to cover them.

Table 3.2 summarizes the availability of server and client implementations of various directory services.

Service	server platform	client platform
Bootp/DHCP	UNIX, Windows	UNIX, Windows
NetBIOS/WINS	UNIX, Windows	Windows
DNS	UNIX, Windows	UNIX, Windows
Hesiod	UNIX, Windows	UNIX
NIS	UNIX	UNIX
NIS+	UNIX	UNIX
LDAP	UNIX, Windows	UNIX
ActiveDirectory	Windows	UNIX, Windows
NDS	UNIX, Windows	UNIX, Windows

Table 3.2: Availability on different platforms

Bootp/DHCP server and client implementations exist for both UNIX and Windows. Windows has the DHCP service built-in. There are UNIX systems that come with Bootp/DHCP support bundled with the operating system but there is a freely available implementation made by the Internet Software Consortium (ISC) that can be used on all recent UNIX variants. There are hardware that supports the Bootp protocol even before the operating system is loaded; this support makes it possible to boot the operating system from the network instead of a local device. That's how diskless systems work. Even ordinary PCs can be made work as diskless workstations with the addition of a boot EPROM. Nearly all network cards has a socket for inserting such an EPROM module to be used during boot, and there is a free boot EPROM building kit available called Etherboot.

NetBIOS is the protocol of the DOS/Windows era. Recent Windows versions can run completely over TCP instead of NetBIOS but the WINS service is still essential for resource sharing and browsing. For the UNIX world, there is an open source implementation available in the Samba package. It can act as either a server for Windows systems or as a client for accessing Windows shares. In theory Samba could be used between two UNIX systems to share data but it is rarely used that way.

The most commonly used directory service nowadays is undoubtedly the DNS. If a computer claims Internet support, then it has at least basic support for the DNS too. Server implementations exist for nearly all operating systems. The Hesiod system that was implemented on top of the DNS has much smaller popularity. It was designed to work on UNIX-like systems only but it did not get very popular. There are some Hesiod configurations in use today but their number is decreasing. It is unlikely that newly emerging platforms will have Hesiod support in the future.

NIS and NIS+ both belong to the UNIX world. NIS being the older and simpler got wide acceptance, nearly all existing UNIX variants support it. There are differences between implementations since NIS is not standardized but such differences usually does not harm interoperability (it is possible that some more advanced features could not be used in a diverse environment, but the basic functionality should work). NIS+ does not have such a wide range of support. NIS+ client implementations exist for Solaris, Linux, AIX and FreeBSD; server implementations exist for Solaris, AIX and BSD. IBM has licensed the NIS+ code that come with Solaris 2.5 and included it in its AIX operating system. They did not update the documentation properly so the IBM NIS+ documentation still has a reference to Solaris 2.5...

The most painful thing about NIS+ is the lack of server support for Linux. Although the FreeBSD server implementation can be compiled with some effort, it is known to have problems and is not suitable for production use. The other shortcoming is the security part: only the NIS+ version included in Solaris 7 or higher has security measures that are considered sufficiently strong (see section 3.2.5). And if the strong security mode is enabled on the server, older clients (such as the implementations for other operating systems) can no longer use it.

LDAP servers are available for both Windows and UNIX platforms. ActiveDirectory and NDS also has LDAP support. For UNIX systems, [RFC2307] defines a suggested schema to use for providing network information services via LDAP instead of NIS or NIS+. Support for information retrieval based on this schema is available for several operating systems such as Linux, Solaris, AIX and so on. Since LDAP is becoming more and more popular it is expected that major OS vendors will have some kind of support for it in the near future.

Microsoft's marketing information claims full LDAPv3 support in ActiveDirectory which is actually not the case. The schema ActiveDirectory currently uses is not compatible with the one defined in the LDAPv3 standard ([RFC2256]) so UNIX clients may have difficulty when using ActiveDirectory as an LDAP server. The opposite direction is even worse: since several Windows services has not been converted to use the LDAP protocol but talk to ActiveDirectory using the old Microsoft IPC mechanisms, it is not possible to use a generic LDAP server instead of ActiveDirectory. There are problems with authentication too: ActiveDirectory includes Kerberos support but Microsoft has added its own extensions to the Kerberos protocol so Windows clients can not use ActiveDirectory objects if they authenticate using a third party KDC instead of the one built into ActiveDirectory.

The main problem with LDAP-compatible directory services is already mentioned in the previous section: there is no standard replication protocol so inte-

grating directory service products from different vendors can be difficult. If the currently proposed LDUP protocol gets accepted as a standard and vendors start implement it (NDS already claims conformance to the published draft) the situation may change. ActiveDirectory still has to be changed to be compatible with the existing LDAPv3 standards though.

3.2.5 Authentication, access control, security

Table 3.3 shows the security level of some directory services. It refers to the **maximum** security level achievable by the given service; misconfiguration may result in lower security levels of course.

Bootp/DHCP was designed to be used during the boot process when no configuration information is known on the client (providing such information is the purpose of Bootp/DHCP) so it has very little authentication. The only information that can be used to authenticate a client is the client's hardware ethernet address. This address is normally stored in the network interface controller (NIC) of the client but it can usually be modified from software so it cannot be really trusted. Lack of security is not considered to be a real a problem since information provided by the Bootp/DHCP service is usually publicly known anyway (at least to people who has enough knowledge to interpret it).

The WINS service from NetBIOS was designed to work in a cooperative environment on a small local network so it has little security measures. The protocol has some protection against multiple clients registering the same name but a careful attacker can cause problems by sending bogus name registrations to the WINS server.

Service	auth. method	access control	security level
Bootp/DHCP	hardware	poor	low
NetBIOS/WINS	none	none	low
DNS/Hesiod	none	minimal	low
NIS	none	minimal	low
NIS+	RPC/AUTH_DH	good	medium
LDAP	Kerberos V	good	high
LDAP	X.509	good	high

Table 3.3: Directory security

NIS uses the Remote Procedure Call (RPC) protocol between the client and the server. RPC has multiple authentication modes: no authentication, UNIX authentication (the client tells the server who he is and the server beleives it), and DES authentication (based on Diffie-Hellman key exchange; more on this later). NIS uses RPC with no authentication. At the time of its creation, this was considered secure: the server does not modify the maps directly and the sources of the maps are world readable anyway. The problem with this is that this does not consider se-

curity at the client side. Since the server is not authenticated, an attacker who can watch the communication between the client and the server may be able to send fake results to the client (see figure 3.7). Since most NIS/RPC implementations use the UDP protocol such an attack is considered easy nowadays. More on this topic can be found in [NISSEC]. It does not help even if the RPC layer uses TCP since there are existing techniques today for injecting false packets to live TCP connections.

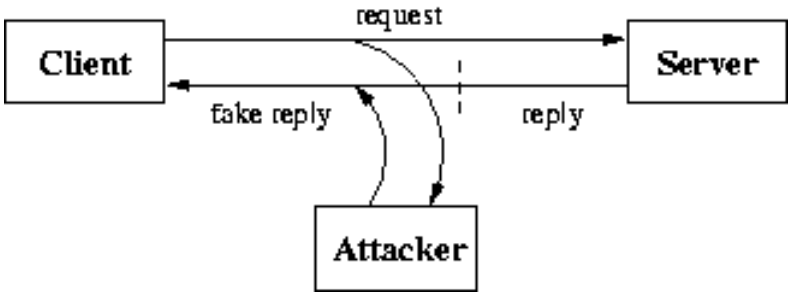

Figure 3.7: Sending a forged reply to a client

Lack of authentication can also be a problem for Bootp/DHCP too: since clients do broadcast queries and use the answer from the fastest server, it is possible to set up a fake server which answers faster than the original. Such a server can force the clients to download a hacked bootimage that installs backdoors in the operating system, configure a fake gateway that can therefore monitor all traffic before bouncing it to the real gateway, or just to annoy users by sending bogus configurations that prevent their system to work correctly. There are no good protection against these problems so one must think carefully about how much a system configured by Bootp/DHCP can be trusted when creating a Bootp/DHCP setup.

DNS/Hesiod has basically the same level of security as NIS. The base DNS protocol does not have any security measures and it uses UDP in almost all implementations. Attacks against DNS service are not uncommon. As a side note, the weakness of DNS security is the cause that hostname-based authentication methods (such as used by the traditional `rsh`, `rcp` etc. utilities) considered insecure nowadays. There are ongoing development for adding cryptographic signatures to DNS messages, but it is not widely used yet.

NIS+ is similar to NIS in the regard that it uses the RPC protocol. But contrary to NIS, NIS+ uses the `AUTH_DES` (or `AUTH_DH`) RPC authentication flavor. `AUTH_DES` uses a 192-bit Diffie-Hellman key exchange protocol for negotiating a *session key*; the session key is then used to add integrity protection using simple DES encoding. The problem is obvious: when the `AUTH_DES` method was defined back in 1989, these algorithms seemed to be strong enough. But time has passed, and a 192-bit key is considered very weak nowadays (the recommended size nor-

mally 1024 or higher), and DES is known to be breakable using brute-force in quite a short time (what is "short" depends on how much money and computing power can be thrown in; it can be between a couple of minutes to a couple of weeks). Seeing the problem Sun come out a solution in their Solaris 7 operating system. Using the newly defined `RPCSEC_GSS` RPC authentication mechanism (based on the GSS-API specification) they provided a way to use 640 and 1024 bit Diffie-Hellman keys. The problem with this approach is that it is not backwards-compatible so older clients can not interoperate with servers where stronger encryption is enabled. Also, other vendors providing NIS+ implementation (e.g. Linux, AIX) has no support for it. Also, the usage of simple DES remained, so if someone has the computation power to brute-force a DES key within the lifetime of a session key, the authentication process can be compromised.

Version 3 of the LDAP protocol defines two possible authentication methods: simple password based authentication, and SASL authentication. Since SASL is an abstract protocol definition, nearly every sane authentication systems can be used with LDAP-based systems. The most commonly used ones are CRAM-MD5 (simple digest-based method to avoid plain text passwords), GSSAPI/Kerberos V, and X.509. OpenLDAP allows the use of all of these methods through the usage of the Cyrus SASL library (which is the most commonly used open source SASL implementation).

NDS currently supports simple password-based authentication; the protection from network sniffing can be achieved by using TLS to encrypt the communication between the client and the server. Novell also offers an extension module for NDS which provides RADIUS authentication support.

ActiveDirectory adopted Kerberos to be its primary authentication protocol. This is a good thing at the first sight but there are of course problems. Microsoft extended the Kerberos protocol with proprietary authorization information which results in clients authenticated by other KDC implementations not being able to use ActiveDirectory resources. The other problem is that the Microsoft version of Kerberos uses an encryption algorithm which is not standard and Microsoft does not allow other vendors to implement it.

After authentication, the next thing that needs attention is access control. DNS/Hesiod and NIS has very limited capabilities in this field: they can grant/deny access based on the client IP only. NIS+ has very good access control semantics: specific rights (create, delete, modify etc.) can be granted on directories, on tables, on row of tables and on coloumns of tables (see also section 3.3.5). Understanding the interaction of these privileges may not seem to be easy for the first sight, but they can be used to provide very fine-grained access control.

The LDAP protocol definition does not define access control. Such a specification is currently under development but has not been finalized yet. Despite the lack of a standard, every LDAP implementation provide some means of access control but they differ in both their syntax and application range. For example, OpenLDAP (an open source LDAP implementation) has very robust ACL syntax: access rights can be specified based on object name (DN), attribute types and attribute values.

Access can be granted or denied according to the client's IP, authenticated identity, group membership and so on. In the case of OpenLDAP, the ACLs are defined in the server's configuration file so they are static. There is support for dynamic ACLs called ACIs, but they are not standard yet so their syntax and semantics are subject to change.

NDS has similar access control semantics as OpenLDAP but it stores ACLs entirely in the directory itself. It has the advantage that ACLs are easy to modify on-the-fly without the need to restart the server. The disadvantage is that when calculating the effective rights of a client the server has to traverse the directory tree where the object representing the specific client lives and it results in performance penalty. However, such a performance penalty is often not noticable. The other problem with storing ACLs in the directory is that it can be hard to maintain them. When an object has to be granted specific rights in several different parts of the tree, it can be hard to oversee what rights are in effect. Although questions like "what rights does user Joe has" can be answered by performing a search operation on the attributes holding access control information (and indeed the NDS user interface has support for such queries), questions like "who has been granted some kind of access" can be hard to answer.

ActiveDirectory follows NDS and also stores access control information in the directory.

3.2.6 Administration

In the real life it is not enough to have a directory service that can handle our data, survives errors and serves all clients fast. The directory service has to be administered and it is very important how easy or difficult that task can be. Some directory services has more or less standard administrative tools but for large networks these tools can turn out to be not optimal and proprietary tools has to be created. If this is the case it is important that there should be a well-defined API for accessing and manipulating the directory.

Table 3.4 summarizes the availability of standard command line tools, GUI and API. For the tools a "yes" means that there are a more or less standard set of tools or GUI applications that can be used to manipulate the directory. A "no" does not neccessarily mean that there are no such tools but it may indicate that they are not standard and not used widely.

Bootp/DHCP servers work using a static configuration which has to be changed externally. The simplicity of the protocol does not make it neccessary to have dedicated tools or API. Of course the Windows implementation has graphical interface but it is still rather simple.

WINS is meant to work completely automatically so it has no direct access interfaces at all.

NIS has a master source of information maintained in traditional ASCII format. The NIS framework uses standard UNIX tools such as make and some own utilities to convert these ASCII files to database files used by the server processes.

Service	command line tools	GUI tools	standard API
Bootp/DHCP	no	no	no
NetBIOS/WINS	no	no	no
DNS	yes	no	yes
Hesiod	no	no	yes
NIS	yes	no	no
NIS+	yes	no	yes
LDAP	yes	yes	yes
ActiveDirectory	yes	yes	yes
NDS	yes	yes	yes

Table 3.4: Administration interfaces

There are also some tools for querying the maps from the command line mainly for debugging purposes. Apart from these there are no dedicated management utilities available for NIS but the source data on the master server can be manipulated by whatever software is capable to edit traditional UNIX configuration files.

For NIS+ the things are better. There are good command-line tools available for manipulating every aspect of the database. With the help of scripting languages these tools can be used to build quite sophisticated administration tools. There is also an API defined for manipulating the database from programs written in C. For Solaris, Sun created the Solstice utilities which provide a graphical interface to NIS+ as well. It has one big problem only is that it is not easily extensible so if there are non-standard tables being utilized to hold other information about users or computers, Solstice can not deal with them.

The DNS was designed to work from static configuration files just like Bootp/DHCP. There are a couple tools for querying the database from the command line (like `dig`, `host` or `nslookup`) which can be found on almost every system and can be considered quasi standard. The same goes for API: most UNIX-like systems integrated the resolver library from the BIND distribution.

When designing Hesiod, MIT decided not to include administration facilities. They did it because they wanted to use the Athena Service Management System (SMS) to accomplish this task. Hesiod does not depend on the availability of SMS but without SMS, the Hesiod data base is just a set of text files in BIND resource record format that must be managed with a text editor. This design has the advantage that small sites does not have to install the Athena management system to use Hesiod while large sites can enjoy the advantages of having good management tools.

For LDAP, there are quasi-standard command line tools originating from the first implementation made at the University of Michigan. There are a good number of both free and commercial GUI based management tools for LDAP. Such tools are often suffer from two problems: either they are too general that one can only deal with low-level information at the object level (e.g. when you want to create

a new user, you have to know what object classes does it need to be in, what attributes must it has etc.), or provide high-level abstraction but only support a limited range of directory schemas and layouts. There are some promising tries to overcome these limitations but the Swiss Army knife of LDAP GUI applications has not been written yet.

Commercial directory implementations concetrate much more on the user interfaces and it is true for both NDS and ActiveDirectory. NDS has a quite good GUI interface called ConsoleOne. Since NDS is on the market for about 8 years now, Novell had the time to develop an administrative interface that is efficient and can solve the problems arising in most directory deployments.

ActiveDirectory follows the Microsoft traditions to offer a good looking GUI environment which is not neccessarily the best but can be used to accomplish the management tasks needed to administer a large Windows-based network environment. Problems may arise if interoperability with other, non-ActiveDirectory systems is desired but they are not originate from the limits of the GUI only but rather than the limitations of the directory itself and from the standards not being followed.

3.2.7 Integration with applications

The directory services designed specifically for helping network and operating system management usually do not have too much applications outside their own administration tools. Among these services only NIS+ is flexible enough to be used for more general purposes. One such purpose can be to store additional data about users and computers that are not directly needed for operating them but may provide help to administrative software and personnel. Such information can include personal identification and account status for users and location, identity and reachability of administrator for hosts. NIS+ provides both tools and an API for these information to be used from custom management software.

From the applications viewpoint X.500 based directory services has far more importance. These directory services provide the feature that they can store nearly arbitrary data arranged in a hierarchical structure. There are a large number of applications that provide and use both white pages and yellow pages services using X.500-based directories. Currently LDAP is the most widely accepted and used protocol for communicating in a multi-vendor environment and the importance of LDAP importance is expected to rise even more in the future.

3.2.8 Summary

During the development of computers and the Internet several directory services was created to be used in different environments. The direction was (and still is) to move from simple and specialized services to more generic ones. However there are areas (like boot protocols) where using complex services is not appropriate either because the available resources (like ROM sizes) are limited or simply because

the raw amount of data to manage and the number of requests does not allow the luxury of wasting resources using too generic services (this is the case with the DNS).

In these days both hardware and software platforms change rapidly but there are always old legacy systems that cannot be replaced for various reasons and need to be supported. This can also prevent using new techniques instead of old ones (like using LDAP instead of NIS) but in most cases there are solutions to emulate an old interface using the new service (like using a NIS-gateway backed by an LDAP-based directory).

While other directory systems are not going away in the near future, the visible trends in both the operating system and in the application industry show the growing importance of X.500 based directory solutions. While the original X.500 standard was anything but successful (it had very few real implementations and uses), its effects are enormous. The most successful technologies based on the X.500 standards are clearly the LDAP protocol and the various PKI architectures (see the next chapter). The usage of the LDAP protocol created the possibility to create and efficiently manage both network environments and directory-based information systems to a global scale which was not possible with previous architectures. Of course there are a wide range of problems that still need to be solved and the solution will likely require the modification or even redesign of existing standards and technologies, but the direction has been set.

3.3 Authentication services

3.3.1 Design goals and basic principles

The most traditional authentication mechanism is password authentication: the client sends its identity (e.g. username) and a secret password to the server, which then verifies using a local database that the given identity and password match. There are several problems with this approach:

- Network sniffing. An attacker who can monitor the traffic on the network is able to interpret authentication messages and extract the username and password pairs. Such attacks are very easy and there are even automated tools built for this purpose.

- Fake servers. If a server goes down for some reason (like maintenance) a malicious user might be able to set up a fake server which accepts authentication requests and stores username/password pairs. The client has no way to know if it talks to the real server or not. This method can work even if the network is physically secure from sniffing.

- Administration costs. In a large network, maintaining consistent user and password databases on each server machine can be hard or even impossible.

- Convinience. Most users do not like to enter passwords all the time when they want to access network resources. So it is important to have the possibility of single sign-on when a password (or other secret) needs to be presented only once in a working session and all further authentication happen transparently.

When developed as part of the Athena project at MIT, Kerberos was designed to work in a hostile network environment, to provide strong, mutual authentication and to be able to serve a very large number of users.

Main design goals of Kerberos:

- Two-way (mutual) authentication. That is, not only the service learns with confidence who the client is, but the client, if it wishes, can also be certain that the correct service is being used.
- No clear text passwords should be transmitted over the network
- No clear text passwords should be stored on servers
- At the clients, clear text passwords should be handled for the shortest time possible and then destroyed
- Any authentication compromisations should be confined to the current session or the current user
- Network authentication should go on largely unnoticed in most cases
- Minimize the effort needed to alter existing network services that previously used other kinds of authentication

Kerberos makes some assumptions about the environment it operates in:

- There will be both public and private workstations. Public workstations have little or no physical security
- Diverse network environment with no link encryption. Some or all parts of the network may be vulnerable to active or passive security attacks
- Centrally-operated servers have moderate physical security and known legitimate hardware
- A small number of servers, like the Kerberos authentication servers, operate under considerable physical security
- It is assumed that the clocks of all computers running Kerberized services are loosely synchronized within a few minutes

Kerberos uses secret key infrastructure to provide authentication service. This requires the presence of central trusted servers that store the secret keys of all participants and perform operations that needed for authenticating those participants to each other. The Kerberos model provides a hierarchical namespace divided to *realms*. Kerberos supports authentication accross real boundaries (inter-realm trust) using special principals. These principals has to be manually created in both realms.

Public key cryptography has the advantage that two parties can communicate with each other without the need of a third-party. This feature made it very attractive to be used in security systems. But there are problems which can be solved only by building a complex infrastructure. The first problem is the need to be certain that a public key really belongs to the entity who I think I got it from. If this cannot be verified then the security could be easily circumvented by a classical man-in-the-middle attack. One way to prevent this is to use external channels (like personal meetings) to verify the connection between principals and their public key. This is the method how PGP's web of trust works. For large scale systems such ad-hoc methods are not adequate. To overcome the problem trusted *Certificate Authorities* (CA) was introduced. These authorities can issue certificates that contain both the identity and the public key of the requestor and are digitally signed by the CA. So if both parties in a communication trusts a given CA they can validate each other's certificate using the CA's public key. A certificate can have more than one signatures from different CAs.

Certificates have a timestamp from when they are valid and have a limited lifetime. A certificate can only be used within this lifetime; when it expires, a new certificate should be requested. Certificates can be revoked before they are expire if the private key belonging to the certificate gets lost or is compromised. The procedure needed for revoking is the Achilles heel of the PKI system since if two parties want to communicate there is no way to know if either party's certificate was previously revoked without the need to communicate with a third-party. The list of revoked certificates (*Certificate Revocation List*, CRL) is maintained by each CA. So when checking the validity of a certificate the issuing CA should be contacted to check if the certificate is valid or not. CRLs are normally issued periodically and cached at many places to ensure their availability. When periodic updates are not enough (such at with high-value fund transfers or large stock trades) a more direct status report may be neccessary; this can be achieved by using the *Online Certificate Status Protocol* (OCSP, [RFC2560]) if the CA supports it.

The other major problem besides certificate verification is the storage of private keys. While certificates containing the public keys can be - as their name say - publicly distributed using LDAP directories, FTP transfers or the World Wide Web, secure keys need to be heavily protected. If secure keys can be tampered with then the whole PKI architecture is undermined. The big problem that contrary to password-based systems, private keys are very hard to be memorized by humans and most people would simply refuse to do so if she was asked to. There is no good general solution for the storage of private keys, but there are some possibilities:

- Encrypt them with a symmetric algorithm using a key derived from a password. It has the advantage that passwords can be remembered by humans and the encrypted key can be stored on insecure medium (like a directory service). It has the big disadvantage that passwords are usually way weaker than the secret key itself so the security of the whole PKI is reduced to the level of password security. At places where PKI was introduced because password based security is not considered enough this is clearly unacceptable.

- Store the secret key on a secure hardware medium like a smartcard. This has the advantage of maintaining the security level provided by the PKI's cryptosystem. The disadvantage is the cost of building the hardware infrastructure needed to create and read the cards. Other disadvantage is that a smartcard can be stolen more easily than a password.

- Combine the above two. Store the key on a smartcard but protect it with a password (usually a PIN code). This lessens the dangers of a stolen smartcard. Smartcards can be made to perform all neccessary crypto operations needed by the PKI themselves thus eliminating the need of reading out the secret key from the smartcard. Using this technique it can be practically impossible for an attacker to stole the secret key of some other user.

The basic format of certificates used by the major PKI implementations was defined in [X.509] as part of the ITU/ISO directory standardization process. Since X.509 is part of the X.500 directory standards, it uses the convention of X.500 for naming principals. For this reason the IETF choosen LDAP (which also uses the X.500 naming scheme) as one of the operational protocol for storing certificates and CRLs.

Managing dispersed serial line and modem pools for large numbers of users can create the need for significant administrative support. Both TACACS+ and RADIUS was designed to be authentication systems primarily for dial-up servers and routers. TACACS+ can also function as an authorization and accounting service; accounting support has been also added to RADIUS in later versions.

The overall design goal of TACACS+ is to define a standard method for managing dissimilar Network Access Servers (NASs) from a single set of management services such as a database. A NAS provides connections to a single user, to a network, or subnet, and interconnected networks.

RADIUS is somewhat similar to TACACS+ but while TACACS+ is the product of one corporation (Cisco) RADIUS is more widely supported. RADIUS has the disadvantage that authentication and accounting cannot be separated so nicely as in TACACS+: TACACS+ allows you to put authentication, authorization and accounting to different servers or even change some of them to non-TACACS+ services (like changing the authentication method to Kerberos but still using TACACS+ for authorization and accounting).

GSSAPI was designed to be a generic way to talk to different authentication systems. Using GSSAPI eliminates the direct dependency of an application on a particular security service which gives the benefits of smaller and cleaner code to the programmer and the ability to freely choose the authentication system to the administrator. GSSAPI provides authentication, data integrity and confidentality services in a generic manner independently of the underlying security mechanism. GSSAPI is also independent from the protocol environment and association meaning that GSSAPI invocations can be embedded in a given protocol hidden from the caller but the application may also call GSSAPI services directly without the fear of conflict with the protocol layers it uses.

The SASL protocol was designed to provide an easy way to add authentication, integrity and confidentality support to connection-oriented protocols. Using SASL the upper level protocol can identify and authenticate an user to a server and negotiate further protection of protocol interactions. Once negotiated, a security layer can be inserted between the protocol and the connection.

SASL can use several security mechanisms. The server may advertise the set of supported SASL mechanisms and clients can choose the one they wish to use. After the mechanism selection, there is a series of server challange - client response messages specific to the selected security mechanism. During these messages the security mechanism performs authentication, transmits an authorization identity and negotiates the further use of a mechanism-specific security layer. If such a layer is negotiated, it is applied to all subsequent data following the authentication phase.

SASL can use the GSSAPI security mechanism. With this combination both the interface for talking to the used authentication service and the inclusion of the security provided by that service in existing protocols becomes easy. This way applications and protocol implementations can exploit the full power of GSSAPI-compliant security services without the need to explicitly depend on such a service.

3.3.2 Quality of Service

Kerberos was designed to be a distributed authentication service. The protocol described in [RFC1510] does not define authorization, integrity or confidentality features. However, the secret session key used for authentication can also be used for exchanging safe (integrity checked) and private (encrypted) messages. The Kerberos V GSSAPI mechanism defined in [RFC1964] can be used if standardized methods for providing integrity and/or confidentality are needed. It is generally a good idea to use GSSAPI instead of the API of a specific authentication service anyway.

PKI can be used for authentication, digital signatures (integrity checking) and non-repudiation purposes. Non-repudiation is a property achieved through cryptographic methods which prevents an individual or entity from denying having performed a particular action related to data. It is very important in business applications when certain actions have financial consequences.

Various PKI implementations usually offer the possibility of data encryption as well. However, encryption is not done directly by public key cryptography because it is usually very resource-consuming. Instead, the involved parties agree on a common secret key using public key methods and then use some symmetric encryption algorithm with this negotiated secret key to actually encrypt the data. Because of this, the strongness of authentication and encryption may be different in a given PKI implementation.

Both the TACACS+ and RADIUS protocols offers authentication only, there is no support for either integrity or confidentality services.

GSSAPI being a generic API layer has a wide range of QoS parameters. A specific authentication service which provides GSSAPI support of course may not implement all of these features. The supported QoS features are:

- Delegation. It enables the transfer of rights of the initiator (client) to the acceptor (server). Using delegation the server can authenticate itself as an agent of the client
- Mutual authentication
- Replay detection
- Out-of-sequence detection
- Anonymous authentication
- Per-message confidentality protection
- Per-message integrity protection
- Mechanism-dependend quality of protection (such as choosing the strength of encryption)

The only lacking thing is support for non-repudiation. This is usually not a problem because GSSAPI is usually used in the lower levels of protocol stacks where such information is usually not needed.

3.3.3 Underlying algorithms

Kerberos uses symmetric encryption techniques to provide authentication service. When a client wishes to authenticate itself to a server it relies on the *authentication server* (*AS*) to generate a new encryption key (called the *session key*) and distribute it to both parties using a Kerberos *ticket*. The ticket is a certificate issued by an authentication server, encrypted using the server key. Among other information, the ticket contains the random session key that will be used for authentication of the principal to the server, the name of the principal to whom the session key was issued, and an expiration time after which the session key is no longer valid. The

AS sends this ticket to the client who then forwards it to the server as part of the authentication request.

To support single sign-on, some kind of secret has to be cached on the client. To limit the consequences of a possible compromisation of this secret, it should not be the user's password. Kerberos uses a special ticket called the *Ticket Granting Ticket* (*TGT*) for this purpose. Using this technique the user's password needs to be present only for a very short time on the client system. When the client wants to authenticate itself to a new server, it contacts the *Ticket Granting Server* (*TGS*) to do this. The ticket granting exchange is identical to the authentication exchange except that the ticket granting request has embedded within it an authentication request, authenticating the client to the authentication server, and the ticket granting response is encrypted using the session key from the TGT, rather than the user's password. Theoretically the AS and the TGS can be separate services but in practice they are usually implemented by the same daemon.

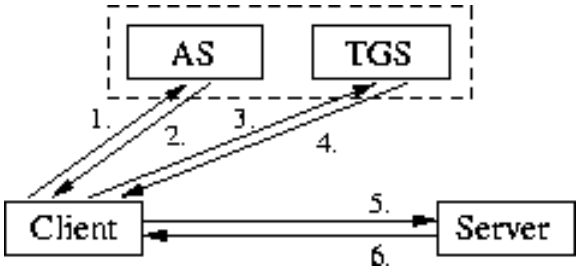

Figure 3.8: The Kerberos authentication protocol

A simple case of the Kerberos authentication protocol shown in figure 3.8 consists of the following steps:

1. Initial request (AS_REQ): contains the client's identity and the identity of the server (usually the TGS) for which it is requesting credentials.

2. The response (AS_REP), contains a ticket for the client to present to the server, and a session key that will be shared by the client and the server. The session key and additional information are encrypted in the client's secret key. The AS_REP message contains information which can be used to detect replays, and to associate it with the message to which it replies.

3. In order to provide single sign-on the client requests a TGT from the TGS (TGS_REQ). The message format for the TGS exchange is almost identical to that for the AS exchange. The primary difference is that encryption and decryption in the TGS exchange does not take place under the client's key. Instead, the session key from the ticket-granting ticket is used.

4. The reply (TGS_REP) sent by the TGS contains the requested credentials, encrypted in the session key from the ticket-granting ticket.

5. When the clients want to authenticate to a server, it sends an application request (AP_REQ). It contains a ticket, an authenticator, and some additional bookkeeping information. The ticket by itself is insufficient to authenticate a client, since tickets are passed across the network in cleartext (tickets contain both an encrypted and unencrypted portion, so cleartext here refers to the entire unit, which can be copied from one message and replayed in another without any cryptographic skill), so the authenticator is used to prevent invalid replay of tickets by proving to the server that the client knows the session key of the ticket and thus is entitled to use it.

6. Typically, a client's request will include both the authentication information and its initial request in the same message, and the server need not explicitly reply to the AP_REQ message. However, if mutual authentication (not only authenticating the client to the server, but also the server to the client) is being performed, an AP_REP message is required in response.

Kerberos is based on symmetric encryption algorithms. [RFC1510] specifies the usage of DES encryption only which is considered rather weak nowadays. There are protocol extension proposals for using *3DES*, *IDEA*, *Rijndael*, *Twofish* and *Serpent* algorithms. 3DES and IDEA are already used in many existing implementations while the others are still in the proposal phase.

For integrity checking, [RFC1510] specifies variants of *CRC32*, *MD4* and *MD5* digest algorithms. Later proposals define the use of *SHA1* ([FIPS180]) and *SHA256* algorithms.

The Kerberos protocol has support for some additional features like renewable tickets (tickets which lifetime can be extended without the need of authenticating again), forwardable tickets (TGTs that can be sent to the server), proxiable tickets and so on.

There are several methods to enchance Kerberos. There is a method called preauthentication by which the client can provide evidence during the initial request that she knows her key before a ticket is issued. This method was meant to make it harder to mount dictionary attacks against the Kerberos service. [TWU] describes such an attack and also how timestamp-based preauthentication can also be circumvented. It offers the possible solution as using the *Secure Remote Password protocol* (*SRP*) developed at the Stanford University ([TWUSRP]). Note that [RFC1510] explicitly states that Kerberos itself does not provide protection from dictionary based attacks.

An other preauthentication method which is actually used by several commercial implementations is the usage of one-time passwords. This technique has the feature that it protects from compromises resulting from stoling passwords using a hacked login program on the client machine. Since the user needs to present a new one-time password every time she wants to authenticate besides her normal password, stealing the normal password is not enough to impersonate the user.

It should also be noted that there are efforts to combine Kerberos with public-key cryptography. It would allow combining the benefits of both worlds: PKI

certificates could be used to request initial tickets while the performance burden of certificate checking and public key cryptography protocols could be eliminated on subsequent authentication sessions.

Public-key cryptography was invented in 1976 by Whitfield Diffie and Martin Hellman, as a key exchange mechanism supporting an otherwise secret-key system ([DH]). The Diffie-Hellman algorithm is based on the difficulty of solving a discrete logarithm problem. The most well-known public key encryption algorithm was invented by Ron Rivest, Adi Shamir, and Leonard Adleman and called *RSA*. The RSA algorithm builds on the hypothesis that the factoring of large numbers is difficult. It is important to note that neither the discrete logarithm nor the factorisation problem is **proved** to be difficult. If someone could come up with a fast solution for these problem it would be a very hard hit for public key cryptography.

The DSA algorithm was created by the National Institute of Standards and Technology (NIST) to be used as a digital signature algorithm as part of the Digital Signature Standard ([FIPS186-2]). Contrary to the RSA algorithm, DSA can only be used for signing and not for encryption. There is a version of DSA based on elliptic curves called ECDSA. The ECDSA algorithm has the advantage that it can use much smaller keys to achieve the same level of security than the original DSA algorithm, making it ideal for small embedded systems (like smartcards).

The public key algorithms currently adopted by PKIX for digital signatures are RSA and DSA. The recommended digest algorithms for signature generation are MD2, MD5 and SHA1. The PKIX certificate specification ([RFC2459]) does not require implementations to use these algorithms, it only specifies that if they do use them then they must do it in a standard way. Implementations are also free to use other algorithms.

Public key cryptography can be used for 4 basic things:

1. Digital signatures (integrity protection)
2. Data encryption (confidentality protection)
3. Non-repudiation
4. Key agreement

The X.509 certificate format allows specifying the allowed usage of a given key. The reasoning behind this is the different usages mean different attack possibilities on a given key. For example, digital signatures produce only a small amount of encrypted data so performing various attacks based on analysis of the encrypted data are much harder than for keys used for encryption, where such data usually available in large quantities. Therefore it is advised to use separate keys for different purposes: the more protected signature keys can be used for a long time while the more vulnerable encryption keys can be changed on a more regular basis. The policy of separate keys for separate purposes may be even enforced by laws or company security policies.

The TACACS+ protocol described in [RFC1492] does not support any encryption of data but the Cisco implementation offers a simple encryption method by calculating an MD5 sum of the session identifier, a secret key shared by the client and the server, the version number, the sequence number and the previous hash (if more than 16 bytes has to be encrypted) and then XORs this hash with the data. If the common secret key is strong enough this should provide adequate protection but the strength of this encoding has not been studied thoroughly (or the results was not published). There are some known weaknesses of the encoding found by Solar Designer (solar@false.com): it has no integrity checking so an attacker can alter certain parts of the communication without notice. This can be used to alter accounting data. Second, there are no protection against replay attacks which is again mostly a problem for the accounting parts. Third, there is a possibility to force reply packets to be encrypted using a session identifier of the attackers choice which makes them vulnerable to crypto analysis attacks. Fourth, due to the small size of the session identifier and the birthday paradox, it is very likely that there will be two sessions encoded with the same session identifier amongst about 100,000 sessions (which means about 20,000 dial-up sessions). Given these weaknesses one may wonder how secure TACACS+ can be.

RADIUS just like TACACS+ is also using a secret shared between the client and the server to provide security. But contrary to TACACS+, this secret is not used to encrypt the whole conversation. Instead only the user-supplied secret information (usually a password) used in authentication packets is encrypted using a similar MD5-based encryption technique than in TACACS+. If the server considers the password to be valid, it responds with a set of configuration information stored in its database for that user. The response packet has integrity checking using an MD5 hash of the data in the response packet, a random number sent by the client as part of the request and the shared secret. An attack against this protocol based on brute force guessing the shared secret was described by Rich Friedeman in a letter sent to the Bugtraq mailing list in 1997.

It should be noted that the semantics of the secret the user sends to the RADIUS client which then forwards it to the RADIUS server is not defined. It can be a conventional password but there are existing implementations to use one-time passwords or hardware based solutions like the SecureID card made by RSA Laboratories.

3.3.4 Scalability, availability

The authentication system Kerberos uses does not require vast amount of resources. Usually a quite small server can perform authentication tasks for a large number of clients. Kerberos can use DNS SRV records to locate KDCs so it is easy to add a new KDC if one cannot cope with the amount of requests. What is a problem that there is no standard replication protocol. The original MIT implementation uses a master-slave setup with read-only replicas, and uses the kprop protocol which simply dumps the whole database on the master and transfers it to the clients. In

most environment it is enough but in an environment with multiple thousands of users it can be a performance bottleneck. Heimdal (an European implementation of the Kerberos V protocol that aims to provide reasonable API compatibility with the MIT implementation) has an experimental protocol called hprop that can be used for incremental replication which results in better performance. ActiveDirectory's Kerberos implementation relies on the ActiveDirectory replication protocol to distribute authentication data to slave servers.

In the case of a large organization the multi-realm support of Kerberos can be utilized to split the database to smaller parts. Using multiple realms also has the benefit that each organizational unit can administer its own users if it is required, but having one central administration body is also possible.

PKIX was designed to be able to scale up to even a global size. Being a general protocol and standard set however means that scalability and availability issues are hard to discuss. There are protocols and recommendations that make it possible to build very fault-tolerant and highly available systems but it is often let to the actual implementation how much does it actually use from the defined architecture. Also client side issues for locating certificate verification services for example are only loosely defined and there is no generally accepted standard (instead there are a set of standards). Commercial PKIX implementations of course usually offer consistent interfaces but there are no guarantees that different implementations can interoperate easily. Of course the basics are standard so some interoperability can be achieved if both parties claim to support the PKIX architecture but using the full range of features offered by either implementation may not be possible. It may be very well true that two different implementations offer the same scalability and availability parameters yet they cannot provide it between each other.

Experience has shown that RADIUS can suffer degraded performance and lost data when used in large scale systems, in part because it does not include provisions for congestion control. There is little information available about the scalability of TACACS+ but because of it similar nature it may suffer from the same congestion problems as RADIUS.

For TACACS+ and RADIUS, clients can usually be configured to try to use more than one servers in the order they are defined in the configuration. This makes it possible to provide basic failover capabilities. However, there is no standard support for synchronizing the database of either different TACACS+ or RADIUS servers so it must be solved by external means (like copying the master configuration file to the other servers regularly).

3.3.5 Authorization

As I have already written before, authorization is a tricky part of the business. The major problem with authorization is that it is nearly impossible to do it in a general manner. Authorization means deciding who can do what. The "who" part is easy, since the authentication service being used determines it. But the "what" part is problematic because it depends on solely the service the client is asking for.

Different services provide different set of available operations.

NIS+ is somewhat special because despite being a directory service it is (at least in the default implementation) tightly integrated with the AUTH_DES RPC authentication mechanism. This gives the advantage that authorization is well-defined: every domain, table and every entry in every table has an owner; domains, tables also have an associated group. Access can be granted to the owner, group, everybody authenticated or not authenticated principal to domains, tables and entries and rows in tables. These possibilities together give a fine-grained authorization system.

Kerberos does not have authorization support just because it was meant to be a general authentication system and authorization cannot be solved generally. However, there are parts in the Kerberos protocol which makes it possible to extend tickets to also carry authorization implementation; few applications make actually use of it. Basically the same goes for PKIX also: the X.509 certificate format does not define authorization, but it can be extended to contain extra information to be used for authorization. The disadvantage is that certificates cannot be changed after they are issued so changing authorization status is only possible by revoking the certificate and issuing a new one.

The TACACS+ protocol contains explicit authorization support. This support was designed with mostly dial-up servers in mind and provide methods for specifying the requested service (ppp, slip, shell etc.), the requested protocol (telnet, http etc.), network interface permissions and similar features. The response for an authorization request can be either allow or deny or a set of features that are allowed instead of the set that was requested. In theory this system could be extended to be used with other applications but there are no such implementations.

RADIUS also supports authorization to some degree. First RADIUS servers are enabled to make authentication decision based on implementation-specific details beside the actual password checking. Such decision might depend on the time of the day, number of concurrent sessions, connection time used in the last week and so on. The protocol's support for authorization is simple: the authentication reply packet can contain configuration information, and there are special attributes defined that actually hold information usable for authorization. It should be noted that technically authorization data is not different from general configuration reply data from the protocol's point of view - only its usage on the client side makes the difference.

SASL has very basic support for authorization: together with the authentication data, it can send an identifier (usually a username) specifying the entity whose behalf the client wishes to act in. It allows proxy applications to authenticate to the shielded service using their own identity but specifying the user who initiated the original requests. Applications supporting SASL can freely decide how they want to use the authorization identity; in the most simple case, they may simply reject the request if the authentication and authorization identifiers are not the same.

The GSSAPI specification leaves authorization support to the underlying security mechanisms. There is generic support only for manipulating principal names

(converting between internal and textual formats and providing comparison functionality). It is totally up to the application to implement its own authorization policy upon these functions.

The conclusion is that authorization support is only possible if the target application area is well known (this is the case with TACACS+ and partially with RADIUS). In other cases only some very basic support can be provided and individual applications need to implement appropriate authorization themselves.

3.3.6 Supported platforms

The original MIT implementation is freely available for UNIX systems. There is an alternative free implementation called Heimdal and there are a large number of commercial implementations available for different systems. Most big companies like IBM, Sun or Cisco support Kerberos V in their products. For Windows, Windows2000 has Kerberos support built into ActiveDirectory. It should be noted that the ActiveDirectory implementation has features not compatible with other implementations so there might be interoperability problems between ActiveDirectory and other Kerberos implementations. The usual case is if a client authenticates using a third-party KDC then it will not be able to use ActiveDirectory resources.

PKIX-related products are available for nearly every possible platform. Their features and the range of supported operations vary widely. The most commonly used PKIX application is the usage of SSL/TLS protected services, most notably the secure HTTP service used for publishing sensitive content on the Web. The second most common application is secure messaging using S/MIME which is also available on nearly all platforms. There are commercial enterprise-level PKI solutions which support the whole range of PKI features. These solutions sometimes use proprietary hardware and software solutions but many on them run on platforms like Windows or commercial UNIX implementations.

TACACS+ was developed by Cisco to be used with Cisco equipment such as routers. There are a couple of freely available TACACS+ server implementations that run on UNIX systems.

There are several free implementations available for RADIUS on UNIX platforms. Since RADIUS became a standard most companies making networking devices capable for authentication (routers, intelligent switches, dial-up servers etc.) support it. In Windows, the Remote Access Service (RAS) is also capable to authenticate dial-in users using RADIUS.

GSSAPI being an API for a specific authentication service is supported on whatever architecture that authentication service supports. The most widely used GSSAPI mechanism is Kerberos V. Both free Kerberos implementations (MIT and Heimdal) have GSSAPI support for UNIX-like operating systems and with some effort they can be compiled on Windows too.

SASL being an abstract protocol layer is even harder to catch than GSSAPI. There is an open source library available from the Carnegie Mellon University called Cyrus SASL. It is available for UNIX platforms and supports several mech-

anisms like GSSAPI, PLAIN or CRAM-MD5. There is also a Java implementation which makes SASL available on every platform where a Java compiler or Java Virtual Machine is available.

3.3.7 Integration with applications

Kerberos being available for quite a long time there are several applications that natively support it. The MIT distribution comes with kerberized versions of the traditional UNIX tools rsh, rcp, telnet and ftp. There are kerberized POP3 servers and clients (Pine, mutt, Eudora) available. Even more applications support Kebreros through the GSSAPI interface. In the UNIX world it should not be a problem to find a kerberized version of any application. For Windows the situation is a little worse as official Kerberos support was only recently introduced with Windows2000 and not many applications utilize it currently. Since Microsoft committed itself to the use of ActiveDirectory which in turn uses Kerberos, the range of kerberized applications for Windows is expected to rise in the near future.

The application support for PKIX is very diverse. Some parts like authentication are widely implemented while other parts like non-repudiation support do not have that many implementations. The most widely used application of PKI techniquies is the *SSL/TLS* protocol. It provides transparent link encryption with optional authentication. The authentication is done by using X.509 certificates while the actual encryption uses symmetric key algorithms. SSL/TLS is used by almost all Internet protocols that need data encryption. The big advantage of SSL/TLS is that it can be built into applications without the need to alter the existing application protocol. In many cases it is even possible not to alter the application at all just put it behind a wrapper that does the SSL/TLS encoding (of course this method prevents the main application from taking benefits of certificate-based authentication).

The other well known application of PKIX techniques is the *S/MIME* standard for signing and encrypting e-mail messages. Most commercial and many free e-mail clients support S/MIME messages. The range of security algorithms may be different, especially older commercial software support low security encryption algorithms only due to the former export regulations in the USA. Free software usually do not have such limitations and they support the strongest available algorithms.

There are a couple of applications with TACACS+ support. There is a free GUI application called Gconfig which can be used to edit the configuration of a TACACS+ server interactively. There is an authentication module for Apache and a PAM module which lets any PAM-aware applications use TACACS+ for verifying passwords and accounting sessions.

There are a wide range of applications supporting RADIUS. As for TACACS+, there is an Apache module for HTTP authentication and a PAM module for generic UNIX authentication using RADIUS. Furthermore applications like OpenLDAP or the Diablo news server also support authenticating via RADIUS on systems where

PAM support is not available.

There are many applications using authentication services through GSSAPI and/or SASL. The IMAPv4 protocol used by advanced e-mail servers and clients has SASL support. There are standard security extensions for the FTP protocol using GSSAPI. SASL support was recently added to the SMTP protocol by which nearly every e-mail is delivered today. SASL is also the standard authentication layer for the LDAPv3 protocol so most directory applications claiming LDAPv3 support also support SASL. There is ongoing work being done to add SASL support for the HTTP protocol as well.

3.3.8 Summary

TACACS+ was designed for a specific purpose where only a limited set of features is required. Also it should be as simple as possible because networking devices with very limited resources have to be able to use them. TACACS+ performs well in the environment it was designed but clearly it should not be considered for more advanced tasks.

RADIUS being more widely accepted than TACACS+ can be used for services that require a simple authentication only and do not want more advanced security features like encryption (either because it is not needed or because they can provide it by their own). RADIUS can be especially handy for companies and organizations managing dial-in users as the users dial-in password can be used for accessing other services as well. While it may not be very secure, it is often adequate for simple tasks.

Kerberos is a proven system for medium to large network environments. If it is configured right and used properly, it provides strong network security for a wide range of applications. When selecting a distributed authentication system to use, Kerberos should be among the first to evaluate.

The PKIX infrastructure provides functions that go beyond the need of secure networking like digital signatures and non-repudiation support. It can be used when security is important not only from the network management but from the legal viewpoint too. Its usability in secure messaging makes PKIX an attractive environment for business applications. The major drawback is that the PKI architecture still evolves, there are many competing standards and it is not clear which one will be the winner on the long run. Also there are management problems that were addressed only recently and there is no proven evidence that they will function as expected if the architecture will be used at country or even global level.

One of the most promising projects today is the integration of Kerberos or similar authentication systems with the PKI architecture. Such an integration has the promise to contain the good features of both world and to help overcome the weaknesses of each by using techniques from the other. Such test systems are already exist but the neccessary standardization processes and wide spread usage are yet to come.

Note that there are work in the other direction too: there is a proposal to add

Kerberos support to TLS. TLS by default uses X.509 certificates for authentication. The proposal suggests using Kerberos tickets instead of the certificates. This can give authentication support to TLS in network environments that have Kerberos installed but do not want to deploy PKI technologies due to their higher installation and maintenance cost.

Given the range of already existing authentication systems and the speed they evolve and new ones emerge, abstract interface and protocol layers like the GSS-API and SASL play a very important role. Application developers can have the feel of stability that the authentication system will not change under them and the knowledge that their products will be usable in a wide range of security environments. System designers and administrators have the freedom of choosing the most appropriate authentication system for a given environment without the restriction that the applications they need to maintain do not support it.

Chapter 4

Case study

4.1 Background

The network environment of the Department of Informatics of the Eötvös Loránd University is a good example as how different directory services change over time. The department has several UNIX servers and a large number of individual workstations. The constantly growing set of services and userbase created the need for efficient central management techniques where a small number of system administrators can manage a large number of users and computers.

Apart from the DNS which is used everywhere, the first directory service in use was NIS in 1996. It was used on a Sun machine acting as the NIS master and there was a couple of Linux machines acting as clients.

In 1998, the system administrators at that time decided to move from NIS to NIS+ in order to take benefits from the improved security and performance of NIS+ over NIS. At that time NIS+ support was only available for Solaris so it was used only on two Sun machines acting as central public servers for the Department. Later in this year I began for testing NIS+ on Linux machines (NIS+ support for Linux was rudimentary and required the use of unstable versions of several critical software). NIS+ support for Linux was stabilized after a couple months of time.

For administering our NIS+ environment several tools were created. The latest and most complex was created by me as my big program in 1999. It featured a form-based interface for adding new users, changing their account status and checking the validity of accounts using a central database called ELTEdb managed at the Centre of Information Technology of the University. My management program used non-standard NIS+ tables for storing additional information about the users.

As the number of students grew at a high rate concerns arose in the year of 2000 that a more powerful directory service is needed. NIS+ functioned well but it was hard to use for general purposes so I began searching for a new directory service which could provide better integration with applications and future services. These planned services included white pages for e-mail addresses, electronic phone book and PKI certificate repository.

Also, the security provided by NIS+ was becoming insufficient as the computing power of workstations grew. Sun introduced new, stronger security modules for NIS+ in the Solaris 7 operating system but enabling stronger security on the servers would made them incompatible with old clients and there was no Linux support for the new security modules. At the end of 1999 the Department decided to purchase a new central server which was intended to replace the old and overloaded Sun server. The new machine turned to be an IBM H70 running AIX, and AIX had no support for strong NIS+ security either.

4.2 The choice

Considering the problems mentioned in the previous section I started to search for a new directory and authentication system to be used at the Department. The main aspects of the search were:

Compatibility. The authentication and the directory service should work together seamlessly. We did not need a separate authentication and a separate directory service, we needed the system where the two are highly integrated.

Scalability. The Department already had about 2000 users and this number was expected to grow. Both the directory and authentication system should be able to deal with this amount of data. Also, there were plans to use the same directory and authentication service in public laboratories so the system should be able to deal with a couple hundred client machines.

Reliability. Since several very important services like e-mail delivery or Web service are dependant on the directory service it should be robust and fail safe. It should be able to run distributed on multiple hosts and provide automatic failover if one of the server machines goes down.

Security. As mentioned above we were planning to use the directory and authentication system in public laboratories where the security of individual hosts is considered low. The authentication service should provide protection about network sniffing. Also, the possibility of inserting backdoors to the operating system used on laboratory machines should be considered; the authentication system should be able to limit the consequences of such hostile actions.

Single sign-on. It was desired that the authentication service should provide support for single sign-on as it is a convinient feature for users.

Application support. As was mentioned above there were plans for services relying on the availability of a directory service. So the system to be choosen should be easy to integrate with other applications.

Support for other platforms. Although we were thinking about supporting only 3 flavors of UNIX-like operating systems (namely AIX, Linux and Solaris)

the directory and authentication system should provide chances for supporting Windows machines later.

Open source implementation. Due to the high costs of the new central server it was unlikely that the Department would spend money on either an authentication or directory service in the near future no matter how many new features would it offer. Linux support for commercial systems is often limited and requires the use of specific kernel or C library versions. Commercial systems may not support other architectures which can be a problem if we want to move to a different platform in the future. Also I was expecting various problems during the implementation phase and closed source software has the big problem that good support is usually available at a high cost only. These reasons together meant that commercial software was not the way to go, an open source solution was needed for both the directory and for the authentication service.

In the case of the directory service the selection was easy. Looking at the available solutions LDAP was the only one that provided all the functionality we needed. There were an open-source implementation called OpenLDAP that provided the following features:

- Being open source means that OpenLDAP can be built on nearly all recent UNIX-like operating systems. This gives independence from the platforms used at the moment.
- LDAP databases was designed to contain a very large number of entries. Installations with about 100,000 entries were demonstrated to be working which ensured that the Department won't outgrow OpenLDAP in the foreseeable future.
- OpenLDAP has replication support so fail-safe service can be provided. The OpenLDAP libraries provide automatic failover if one of the servers goes down.
- OpenLDAP has an access control checking system which can be used to provide fine-grained access control over the information stored in the DIT. Using ACLs it is possible to enable users to manage some attributes of their entries by their own while other entries are editable (or even visible) only by a group of administrators. The administrator group itself can be defined in the DIT which makes administration easy if people come or go.
- With the release of Windows 2000 with ActiveDirectory support there is a chance that Windows systems will be able to use the OpenLDAP directory. Currently there are unresolved issues created by Microsoft by abusing the existing standards but they might be resolvable in the future.

- As LDAP becoming more and more popular there are several applications that can benefit from it. These applications involve e-mail systems (both for mail delivery software and for address book service for user agents), Netscape roaming profile support, PKI certificate repository and so on.

Of course OpenLDAP has its own limitations which must also be noted:

- The stable version of OpenLDAP supports read-only replication only. There is experimental support for read-write (or multimaster) replication but it is not considered stable yet. That means that the current implementation has a single point of failure but it may be removed in the near future.
- ACLs are good, but especially the evaluation of group ACLs can be slow. If the load on the LDAP server increases this issue will need to be addressed. Fortunately, the open source nature of the code makes it easy to perform modifications to speed up the ACL evaluation code.

In the case of authentication systems, two systems had the features we needed: PKI and Kerberos. Other systems like RADIUS did not provide the level of security and integration that I needed.

Kerberos was designed specifically for the purpose which I wanted to use it. PKI had several problems: first, there are multiple competing standards and it is not clear which one will have the most support in the future. The handling of certificates in a PKI architecture is hard and we did not have a good solution for it. And the biggest problem was that PKI is hard to integrate in low-level services such as UNIX login. These problems resulted in choosing Kerberos instead of PKI.

As for the Kerberos implementation, there were two choices: the original reference implementation made by MIT or the European Heimdal implementation. Since at the time of the investigation US export restrictions on crypto software were still in effect Heimdal was choosen.

The features that Kerberos provides:

- It integrates cleanly with OpenLDAP via the SASL/GSSAPI authentication mechanism.
- Kerberos has been proven to support several thousand users. Kerberos authentication servers can be replicated to provide failover and load balancing.
- Kerberos provides adequate security for authentication. The GSSAPI can be used by applications to provide integrity and confidentality support.
- Single sign-on is provided by the protocol by using TGTs.
- Many protocols and applications support GSSAPI authentication which in turn means they can be used with Kerberos.

Problems with Kerberos:

- Heimdal has read-only replication only which introduces a single point of failure.
- [RFC1510] explicitly states that Kerberos does not protect from dictionary attack. In order to be secure one has to use strong passwords which is a problem among users. This problem might be solved by adding SRP support to Heimdal.

4.3 Implementation and migration problems

Choosing OpenLDAP and Heimdal to be used as directory and authentication services did not mean that everything was solved. These services needed to be tested and integrated with already existing applications and operating systems. Also, the existing user database held in NIS+ should have been migrated to LDAP.

I began working on these problems at the beginning of year 2000. At that time OpenLDAP had problems with SASL encryption support: the code responsible for this was a complete mess. I began working on it and came up with a general transport API layer that supported both SASL and/or SSL encryption of data sent over the network. This project lasted for about three months when my changes was accepted and applied to the mainstream version.

Parallel to the coding I began working on a schema that would be used later to represent the users' data. The attributes related to standard UNIX information like `/etc/passwd` was the easy part as they were covered by [RFC2307]. However new attributes were needed for information specific to my management software used for administering users with NIS+. New attributes needed to be defined for student or employment IDs, account creation date, creator of an account, person responsible for an account, account status and so on. Other attributes were needed to provide support for e-mail routing. Currently there are a total of 12 attribute types and 3 object classes defined over the standard ones.

For Solaris and Linux, integrating LDAP with the operating system was relatively easy. These operating systems have a feature called *Name Service Switch* (*NSS*) which enables the dynamic loading of arbitrary modules to be used as information sources for name service calls. Both operating systems support traditional `/etc` files, NIS and NIS+ using the NSS interface. Padl Software has created an open source module called NSS_LDAP which can interface with NSS and provide name service using an LDAP directory as a database.

The problematic operating system was AIX which was used by the yet-to-buy new central server. AIX had no NSS support but it had a somewhat similar interface called Information Retrieval System (IRS). Unfortunately this interface is not completely exported by the operating system: there are support for all databases except the password and the group data. To resolve user and group names, AIX uses a proprietary interface called Security Methods which was not documented by IBM. To get around this, I created a simple translator that could use an IRS module

to retrieve password and group information while acting as a NIS server to the host operating system. Since AIX supported NIS, using this translator it became possible to integrate AIX and LDAP. After the translator was ready, IBM announced the new major version of AIX called AIX5L and made documentation for it available at the IBM web site. Surprisingly the AIX5L documentation had some information about the previously undocumented naming service interface which made it possible to create a native AIX implementation of NSS_LDAP. These changes were immediately accepted to the upstream version and the NIS translator I have written before became obsolote.

Kerberos was not so problematic. However, since I did not consider DES strong enough and 3DES is slow, I have added Blowfish support to Heimdal. For services needing traditional password authentication, I used a PAM module. AIX did not have PAM support by default, but I managed to port the Linux PAM libraries to AIX. The AIX Security Methods interface supports adding new authentication systems so it would be possible to create a Kerberos module for it and use it instead of PAM, but there are a lot more applications supporting PAM than applications supporting the AIX method.

OpenSSH is used in order to enable secure login to AIX. The original OpenSSH does only support Kerberos IV and not Kerberos V, but I found a patch for an earlier OpenSSH version that could be still applied with some minor modifications. The patch supported the old 1.x SSH protocol however; later I added Kerberos support for the 2.x SSH protocol. In March 2001, the IETF published a draft for using GSSAPI with the SSH protocol; I plan to check the availability of patches adding GSSAPI support to OpenSSH in the near future.

Migrating data from NIS+ to LDAP was the easy part. I have written a small Perl script that extracts all data from NIS+ and outputs it in LDIF (LDAP Data Interchange Format) format that could directly fed to the OpenLDAP server. The problematic part was the case of passwords since NIS+ and Kerberos uses completely different encodings. For these reason I created a small program on a server still running NIS+ that could authenticate a user using her NIS+ password and if that was successful, create her Kerberos principal with a password she specified. So every user could log in to the server using NIS+ and set her own Kerberos password which in turn could be used to log in to the new server.

4.4 New services

The first service to make advanced use of LDAP was the e-mail system. Most of the development were done by my colleague called Péter Kelemen. By using an LDAP attribute of users where they wish their mail should be sent, the e-mail system was successfully separated from the home directory service. Before the utilization of LDAP the e-mail service was dependant on the home directory service because mail forwarding information was held in users' directories in traditional `.forward` files. Since I also switched to using an IMAP server instead of local delivery to

traditional mailboxes under `/var/mail`, it is now possible to relocate the entire mail service without users even noticing it if performance or other issues make it neccessary.

Combining LDAP with the e-mail delivery system has other advantages: using the advanced search capabilities of LDAP support of *Full.Name@inf.elte.hu* style e-mail addresses became possible together with the *user@inf.elte.hu* style addresses used so far. This feature is not yet official because a technique for resolving conflicting full names still has to be created.

The next service to move was the Web. Migrating web content from the old server to the new one had some difficulties because of the different ACL syntaxes and semantics of Solaris and AIX. However, a mapping between the two semantics got created by Sándor Bedő and the transition took place. The new Web server uses the same technique as the e-mail system to locate users' home pages. It has the same capability to support *http://people.inf.elte.hu/Full.Name* style URLs but due to the same name conflict problem it is not yet enabled.

I have planned to provide Netscape roaming profile support for all users, but currently this project is suspended because Netscape seems uncapable to use an SSL-protected connection when talking to the LDAP server, and sending passwords over the network in cleartext is not a good idea. I hope that the Mozilla project will solve this issue somewhere in the near future.

E-mail clients with LDAP support can already use the directory for white pages information. The webmail system I deployed can also use LDAP.

There are other services which were planned but not yet implemented. These include the creation of a certificate repository for our users. With some effort, it could be used to provide digital signature service for the Department that is compatible with the recently accepted Hungarian law about digital signatures.

Bibliography

[X.500] *The Directory: Overview of Concepts, Models, and Services*
ITU-T Recommendation X.500
ISO/IEC 9594-1

[X.509] *The Directory: Authentication Framework*
ITU-T Recommendation X.509
ISO/IEC 9594-8

[CHERYL] Cheryl Walton: *LDAP and NDS.*
Netware Connection, November 1999, pp. 18-30
http://www.nwconnection.com/nov.99/ldapn9/index.html

[CHADWICK] David Chadwick: *Understanding X.500 - The Directory*
Chapman & Hall
ISBN 1 85 0322 813
http://www.salford.ac.uk/its024/Version.Web/Contents.htm

[SOLNAM] *Solaris Naming Administration Guide*
806-1387-10
Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, CA, 94303, U.S.A.

[SOLGSS] *GSS-API Programming Guide*
806-3814-10
Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, CA, 94303-4900, U.S.A.

[KRBPLAN] S. Miller, C. Neuman, J. Schiller and J. Saltzer: *"Section E.2.1: Kerberos Authentication and Authorization System"*
M.I.T. Project Athena, Cambridge, Massachusetts
December 21, 1987.

[NEUTSO] B. Clifford Neumann and Theodore Ts'o: *Kerberos: an Authentication Service for Computer Networks*
IEEE Communications Magazine, Vol. 32 (9) pp. 33-38.

September 1994.
ISSN 0163-6804

[BELMER] S. M. Bellovin and M. Merritt: *Limitations of the Kerberos Authenication System*
ACM Computer Communication Review, Vol. 20 (5) pp. 119-132.
October 1990.
ISSN 0146-4833

[TWU] Thomas Wu: *A Real-World Analysis of Kerberos Password Security*
Proceedings of the 1999 Internet Society Network and Distributed System Security Symposium
February 1999.

[TWUSRP] Thomas Wu: *The Secure Remote Password Protocol*
Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium, pp. 97-111.
March 1998.

[KNT] J. T. Kohl, B. C. Neuman, and T. Y. T'so: *The evolution of the Kerberos authentication system*
Distributed Open Systems, pp. 78-94.
IEEE Computer Society Press, 1994.
ISBN 0-8186-4292-0

[NS] R. M. Needham and M. D. Schroeder: *Using Encryption for Authentication in Large Networks of Computers*
Communications of the ACM, Vol. 21 (12), pp. 993-999.
December 1978.
ISSN 0001-0782

[DS] Denning, D., and G. Sacco: *Time stamps in Key Distribution Protocols*
Communications of the ACM, Vol. 24 (8), pp. 533-536.
August 1981.
ISSN 0001-0782

[DH] W. Diffie and M. E. Hellman: *New directions in cryptography.*
IEEE Transactions on Information Theory, Vol. 22 (6), pp. 644-654.
November 1976.
ISSN 0018-9448

[RSA] R. L. Rivest, A. Shamir, and L. Adleman: *A method for obtaining digital signatures and public key cryptosystems.*

Communications of the ACM, Vol. 21 (2), pp. 120-126.
February 1978.
ISSN 0001-0782

[BS] Bruce Schneier: *Applied Cryptography, 2nd Edition*
John Wiley & Sons, 1995.
ISBN 0471117099

[FIPS186-2] *Digital Signature Standard*
Federal Information Processing Standards
Publication (FIPS PUB) 186-2
January 2000.

[FIPS180] *Secure Hash Standard*
Federal Information Processing Standards
Publication (FIPS PUB) 180-1
April 1995.

[ATHENA] George Champine, Daniel Geer, William Ruh: *"Project Athena as a Distributed Computer System"*
IEEE Computer, Vol. 23 (9), pp. 40-51.
September 1990.
ISSN 0018-9162

[HESIOD] Stephen P. Dyer: *The Hesiod Name Server*
Proceedings of the USENIX Winter 1988 Technical Conference
USENIX Association, 1988.

[NISSEC] David K. Hess, David R. Safford, Udo W. Pooch: *A Unix Network Protocol Security Study: Network Information Service*
Texas A&M University
ftp://ftp.cso.uiuc.edu/pub/security/coast/unix/sra/
TAMU/NIS_Paper.ps.gz

[KRBPKI] Matt Hynes: *An Analysis of Distributed Network Security Services: Kerberos and Public Key Infrastructure (PKI)*
Cisco World
http://www.ciscoworldmagazine.com/webpapers/2001/
04_guardent.shtml April 2001.

[RFC951] RFC 951: *BOOTSTRAP PROTOCOL (BOOTP)*
September 1985.
http://www.ietf.org/rfc/rfc951.txt

[RFC1001] RFC 1001: *PROTOCOL STANDARD FOR A NetBIOS SERVICE ON A TCP/UDP TRANSPORT: CONCEPTS AND METHODS*

March 1987.
http://www.ietf.org/rfc/rfc1001.txt

[RFC1034] RFC 1034: *Domain names - concepts and facilities*
November 1987.
http://www.ietf.org/rfc/rfc1034.txt

[RFC1035] RFC 1035: *Domain names - implementation and specification*
November 1987.
http://www.ietf.org/rfc/rfc1035.txt

[RFC1492] RFC 1492: *An Access Control Protocol, Sometimes Called TACACS*
July 1993.
http://www.ietf.org/rfc/rfc1492.txt

[RFC1510] RFC 1510: *The Kerberos Network Authentication Service (V5)*
September 1993.
http://www.ietf.org/rfc/rfc1510.txt

[RFC1964] RFC1964: *The Kerberos Version 5 GSS-API Mechanism.*
June 1996.
http://www.ietf.org/rfc/rfc1964.txt

[RFC2131] RFC 2131: *Dynamic Host Configuration Protocol*
March 1997.
http://www.ietf.org/rfc/rfc2131.txt

[RFC2136] RFC 2136: *Dynamic Updates in the Domain Name System (DNS UPDATE)*
April 1997.
http://www.ietf.org/rfc/rfc2136.txt

[RFC2222] RFC 2222: *Simple Authentication and Security Layer (SASL)*
October 1997.
http://www.ietf.org/rfc/rfc2222.txt

[RFC2251] RFC 2251: *Lightweight Directory Access Protocol (v3)*
December 1997.
http://www.ietf.org/rfc/rfc2251.txt

[RFC2252] RFC 2252: *Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions*
December 1997.
http://www.ietf.org/rfc/rfc2252.txt

[RFC2255] RFC 2255: *The LDAP URL format*
December 1997.
http://www.ietf.org/rfc/rfc2255.txt

[RFC2256] RFC 2256: *A Summary of the X.500(96) User Schema for use with LDAPv3*
December 1997.
http://www.ietf.org/rfc/rfc2256.txt

[RFC2307] RFC 2307: *An Approach for Using LDAP as a Network Information Service*
March 1998.
http://www.ietf.org/rfc/rfc2307.txt

[RFC2459] RFC 2459: *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*
January 1999.
http://www.ietf.org/rfc/rfc2459.txt

[RFC2560] RFC 2560: *X.509 Internet Public Key Infrastructure: Online Certificate Status Protocol - OCSP*
June 1999.
http://www.ietf.org/rfc/rfc2560.txt

[RFC2743] RFC 2743: *Generic Security Service Application Program Interface Version 2, Update 1*
January 2000.
http://www.ietf.org/rfc/rfc2743.txt

[RFC2865] RFC2865: *Remote Authentication Dial In User Service (RADIUS)*
June 2000.
http://www.ietf.org/rfc/rfc2865.txt

Index