Abstract

The general theme for this thesis is computer security. In particular, attention is directed towards secure distributed systems and how role-based access control can be introduced into such systems.

Access control is an important building block for creating secure computer systems. Role-based access control (RBAC) is an efficient way of organizing access control information, both from an administrative and system architectural point of view. To benefit from the advantages of this in a distributed system, several building blocks are necessary. We need a distributed infrastructure that can enforce access control and allows applications to communicate securely. Within the infrastructure, roles and access rights must be managed efficiently. To protect existing investments we must cater for legacy systems and demonstrate how existing applications can be integrated into the distributed infrastructure. This thesis provides contributions in these areas.

An object-oriented architecture for secure distributed systems is presented. The architecture can use role-based access control and allows existing applications to be adapted to the secure environment without need of modification. We also propose a mechanism whereby centralized security management can be introduced into an otherwise distributed system.

The thesis presents a general framework for representing roles and constructing organizational models from roles and their inter-relationships.

We present a design and implementation for introducing role-based access control in already existing systems. In the design, an NFS server is adopted to use role-based access control information. A performance comparison between the original and the enhanced NFS server is included.

The thesis also includes an overview of computer security methods in general and a survey of existing distributed protocols, systems, and frameworks designed with security in mind.

Looking forward to future research, we envision an extended formal framework that includes authorizations and constraints. We also see the need for support-tools that help in the process of translating high-level ac-

cess control policies into explicit assignment of access rights.

Acknowledgments

I'm greatly indebted to a number of persons, without whose support this thesis would not have come into existence.

First of all I want to thank my supervisor and advisor, Nahid Shahmehri. Thanks go to her on many accounts, for talking me into post-graduate studies in the first place, and for coaching, guiding, encouraging, and (occasionally) coercing me into producing this thesis. Thank you!

Thanks go also to all, past and present, members of IISLAB who have helped in creating the friendly atmosphere that makes work so much easier.

I'm very grateful for the valuable comments given by Viiveke Fåk and Patrick Lambrix after having read drafts of the thesis. Thank you!

Of course, incalculable gratitude goes to Camilla, for being there.

The financial support from NUTEK (the Swedish National Board for Industrial and Technical Development), Strategiska Forskningsstiftelsen (the Swedish Foundation for Strategic Research), and ECSEL (Excellence Center in Computer Science and Systems Engineering in Linköping) is gratefully acknowledged.

Contents

Abbreviations

ACI	Access Control Information
ACL	Access Control List
ADF	Access Decision Function
AEF	Access Enforcement Function
APA	Authentication and Privilege Attributes
API	Application Programming Interface
AS	Authentication Server
AUC	Authentication Certificate
CA	Certifying Authority
CORBA	Common Object Request Broker Architecture
DAC	Discretionary Access Control
DES	Data Encryption Standard
ECMA	Formerly, *European Computer Manufacturers Association.* "ECMA" is now the official name of the organization.
ESP	Encapsulating Security Payload
FTP	File Transfer Protocol
GID	Group Identity
GNU	GNU is Not Unix (trademark of the Free Software Foundation)
GSS-API	Generic Security Services Application Programming Interface
HTTP	Hypter-Text Transfer Protocol
IANA	Internet Assigned Numbers Authority
IDEA	International Data Encryption Algorithm
IEC	International Electrotechnical Commission
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	International Organization for Standardization
IT	Information Technology
KDS	Key Distribution Server
MAC	Mandatory Access Control
MAC	Message Authentication Code
MIC	Message Integrity Code

NFS Network File System
NIST The National Institute of Standards and Technology
NNTP Network News Transfer Protocol
OMG Object Management Group
ORB Object Request Broker
PAC Privilege Attribute Certificate
PAS Privilege Attribute Server
RBAC Role-Based Access Control
RDO Role Descriptor Object
RFC Internet Request for Comments
RPC Remote Procedure Call
RSA "Rivest, Shamir, and Adleman" — RSA Inc. is an American software company
S-HTTP Secure-HTTP
SACM Secure Association Context Manager
SCM Security and Context Manager
SDSI Simple Distributed Security Infrastructure
SEAL Software-optimized Encryption Algorithm
SHA Secure Hash Algorithm
SMIB Security Management Information Base
SMTP Simple Mail Transfer Protocol
SPI Security Parameter Index
SPKI Simple Public Key Infrastructure
SSH Secure Shell
SSL Secure Sockets Layer
TCFS Transparent Cryptographic File System
TCP Transport Core Protocol
TGS Ticket-Granting Server
TGT Ticket-Granting Ticket
UDP User Datagram Protocol
UID User Identity
VFS Virtual File System
WWW World Wide Web
XDR External Data Representation

Chapter 1

Introduction

For several years, there has been a clear trend in research and in the market place towards distributed applications and techniques. The days of centralized and monolithic software systems are past. Today, concepts like *client/server*, *the Internet*, *Intranets*, *distribution*, *virtual organizations*, *tele-commuting* and *Cyberspace* are in vogue. These phenomena all have one thing in common, namely the underlying need for *communications*, often taking place over public networks.

Unfortunately, however, communicating information over a network introduces vulnerabilities. For instance, information might get distorted in transit (accidently or on purpose), eavesdropping might result in sensitive information being revealed, or the remote node we are communicating with might in fact be someone other than we think.

Another issue that arises when more and more applications become "distributed" is how to cater for those existing systems that we are loath to do away with, either because they function well today or because they would be very expensive to replace.

Even if a secure means of communications is provided, there are still many other security issues to be considered. In all systems, it is important to manage the use of resources in a secure and efficient way. For reasons of confidentiality, integrity, economy, and others, access to programs, files, printers, etc. in a system is very often subject to *access control*. To get a working access control system, two equally important issues must be considered. There must exist technical solutions that make it possible to put the access control system in place and, equally important, it must be possible to manage and administrate access control information in an efficient and expeditious way.

Consider the following example:

Acme General Hospital has recently invested in a new computer system that among other features allows for electronic storage of patient records and electronic transfer of laboratory test results and X-ray images. In a hospital environment, demands on security are high. Consistency and availability of data must be guaranteed at all times. Ethics as well as legislation also put the demands for confidentiality in focus.

For several years at Acme General there has been a constant personnel shortage due to tight budgets and other factors. As a consequence, doctors, nurses, and secretaries must often move between wards to meet minimal staffing requirements. Short-time extra replacements are not uncommon either. Given the mobility of personnel, the designers of the new computer system realized early on that access control could not be managed on an individual basis. Instead, the designers decided to go for a *role-based* approach.

First of all, the organization of a typical ward was analyzed and a set of *roles* was identified. These were *secretary*, *nurse*, *head nurse*, *physician*, *senior physician* and *head physician*[1]. Secondly, access control in the computer system was tailored to differentiate access to patient records and other resources according to the role(s) a user has. By using this indirect way of expressing permissions it was possible to put in place and verify access control rules without knowing who the individual users of the system would be.

Once a system is in place, it is necessary to allow individual users to take on roles. At Acme General, every employee who needs access to the computer system (and that is almost everyone) is provided with a *personal smart card* that is used together with a pass-phrase to identify the user to the system along the same principles as for automatic teller machines. As every individual is personally accountable for her actions, establishing identity is very important, even though operations in the computer system are performed in the capacity of some role. At Acme, every computer terminal has a card reader into which the user must insert her smart card in order to activate the terminal. To discourage the possible use of someone else's identity, for instance if someone has forgotten a card in the reader, a policy has been implemented that requires users to reassert individual identity when an update transaction is performed.

Due to the strained circumstances of the personnel, the assignments within a ward of individuals to roles vary quite frequently. Before the

[1] This nomenclature corresponds well to the organization of Swedish hospitals. Organizations elsewhere may use other nomenclature.

start of each working week, the *head physician* finalizes a *duty roster* that contains the current assignments, including information about who will be acting as *head physician*.

The scenario illustrates the main idea of *role-based access control*, RBAC. An indirection in the assignment of access rights can greatly reduce the administrative effort needed to maintain consistency in the system. Another important property of RBAC highlighted in the example is that access control assignments can be performed without knowledge of the actual individuals who will fill the roles.

In this thesis we study how role-based access control can be brought to distributed systems. In order to introduce access control in distributed systems, one has to deal with many of the vulnerabilities to communications arising from distribution, as well as other problems. For example, in a distributed system, it is necessary to decide where *access control information* (ACI) should be stored and managed. Another problem to be addressed is how ACI should be retrieved by various security mechanisms.

Figure 1.1 shows schematically the three main logical components in a secure distributed system. A typical situation is where a client wishes to call upon the services of some remote server. In addition to the client and server, somewhere in the system *security information* and *security services* are also provided. This information and these services are needed for ensuring secure communications and for the enforcement of access control.

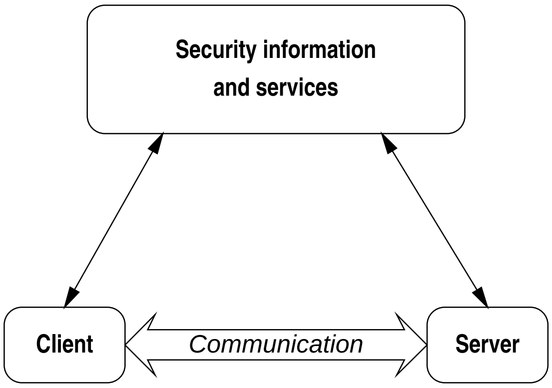

Figure 1.1: Logical components in a secure distributed system.

Contributions

In this work, we elaborate on the simple model in figure 1.1. In bringing role-based access control to distributed systems it is important to provide both theoretical foundations as well as examples of practical applications. This thesis contains several contributions in this direction:

- We present an object-oriented distributed secure system architecture. By using the architecture, existing applications can be integrated into a security framework without major changes. The framework also proposes a way for introducing centralized security monitoring into an otherwise distributed system.
- We propose a framework for modelling organizations using roles and inter-role relations and describe its application to role-based access control.
- We present a design for how role-based access control can be introduced in legacy systems using a modified NFS server. We also present demonstration of feasibility.
- We have performed a survey of existing protocols, systems and architectures designed with security in mind.

To the best of our knowledge, the ideas of introducing centralized monitoring and using an NFS server for role-based access control are novel and have not been presented before.

The structure of this thesis

Chapter 2 places our security-related work into a broader context. The chapter also provides an introduction to security problems found in distributed systems and gives an orientation on some of the building blocks used to address these problems. The chapter is intended to present the necessary foundations for what follows.

Chapters 3, 4, and 5 constitute the main body of our work. These chapters are relatively self-contained and can be read independently of each other.

Chapter 3 contains the object-oriented distributed system architecture. The architecture in this chapter provides an infrastructure for the information flow represented by the arrows in figure 1.1.

The role-description framework can be used to provide some of the security and information services indicated in figure 1.1. The framework is described in chapter 4.

Chapter 5 reports on our work to modify an NFS server to use role-based access control information. The work we present is focused on the server component of figure 1.1.

Chapters 4 and 5 are both slightly modified and extended versions of work we have presented and published earlier [GS96, GDS97].

Chapters 6 and 7 are provided to give the interested reader further background material. Chapter 6 revisits much of what is introduced in chapter 2, but presents a more thorough introduction to computer security. To provide a picture of the current state of the art, chapter 7 contains a survey of existing architectures and systems designed to provide security in distributed environments.

Chapter 8 concludes the thesis by presenting a summary of our main results and by discussing future work.

Chapter 2

Computer security

One dictionary defines "security" as "*The state of being secure*" [McL87]. The same dictionary has several meanings for the word "secure". Most pertinent among them are:

1. *Free from danger, damage, etc*
2. *Free from fear, care, etc*
3. *In safe custody*
4. *Not likely to fail.*

Applied to an individual, these definitions are intuitive and easy to apply. However, when we try to apply them to something more abstract, like an organization, it is not evident how these definitions should be interpreted. If we look more closely, it is clear that in essence what is expressed is an underlying idea of *protection from harm*. Adopting this view, "having security" within the context of an organization comes to mean "having an organization protected from harm".

It is against this background that in this chapter we give an introduction to how the concept of security applies to organizations in general and to computer systems in an organization in particular. The chapter has three sections. First we introduce the *security process* as the chain of events and actions that can bring security to an organization. We then go on to introduce the concept of *computer security* and discuss the complications that can arise when computing and applications become distributed over a network. To show how some of these problems can be solved, the last section introduces some commonly used building blocks that can be used to construct secure systems.

The purpose of this chapter is to introduce the basic ideas and concepts upon which the following chapters are based. For the interested reader, chapter 6 gives a more detailed introduction to computer security.

2.1 The security process

That organizational security should mean protection from harm may seem like a trivial statement. However, stating this simple goal gives us a starting point for initiating a *security process* inside an organization. Ultimately, this process will result in the organization introducing a number of *security measures* that serve the purpose of providing the necessary protection from harm. Access control, the main body our work, belong in this category. It is important to realize there is a long way to go before arriving at specific security measures. For instance, in order to protect from harm, we must first establish what constitutes harm with respect to the organization in question. In the security process, an assessment of risks, threats, and consequences, is used to establish *security policies.* Once established, the security policies govern which *security measures* should be taken. Carrying the security process this far will hopefully protect the organization from harm. However, the state of being secure is not static. External factors that are the basis for the assessment of risks and threats may change over time. Structural change within an organization, due to the security process and other factors, can change the initial conditions under which security assessments were made. To be effective the security process must be continuously reviewed. When countermeasures have been deployed they must be *evaluated* together with current policies. The results from this evaluation are important input when the process starts a new iteration.

Figure 2.1 shows the four main stages in the security process. The first phase in the process is an assessment of which assets there are, what vulnerabilities they have, and what threats there are to the assets. This analysis results in a number of *risks* being identified. Once a set of risks has been identified, work to minimize the risks can begin.

In the second phase, decisions have to be made as to what countermeasures are necessary and justifiable. Here, factors such as cost, complexity and awkwardness of countermeasures must be weighed against organizational requirements and impact of potential damages. Some risks might be considered acceptable while others are deemed grave enough to warrant extensive countermeasures to be installed. We term this phase of strategic decisions *establishment of policies.* There exist a number of methods for carrying out phases one and two, see for instance [Mos92].

Only when overall considerations, economic, temporal, technical and other have been fully carried out will it be known what *countermeasures* should be implemented.

The final phase in the security process is the *evaluation* of current policies and deployed countermeasures. The evaluation should answer questions such as "Do current policies have the desired effect?", "Do our countermeasures implement our policies?", and "How do our policies and countermea-

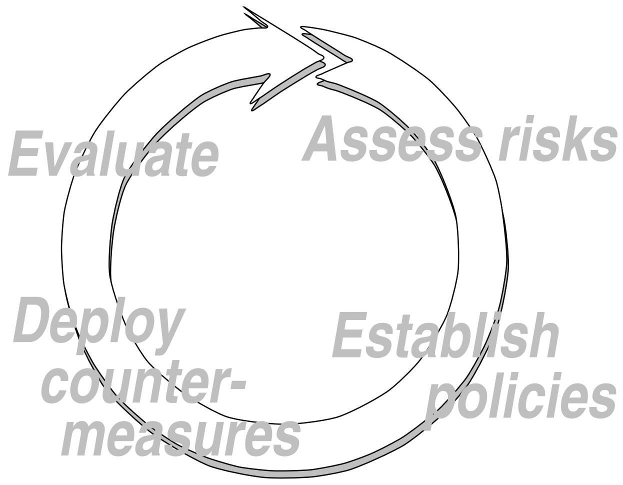

Figure 2.1: The security process.

sures influence our initial assessments?". The answers found in the evaluation phase are important input to the assessment phase of the next process iteration.

Of the phases in the security process, countermeasures and the deployment of countermeasures are probably the areas most studied and best understood. For instance, in [Mos92], Moses has classified countermeasures into seven categories:

Avoidance. The causes of the risk are removed. For instance, dial in access to a network incurs a risk of unauthorized access from unknown sources. This risk can be avoided by not having any modems.

Transfer. The items at risk are moved to a secure place. An example is putting a computer storing sensitive information in a locked room.

Reduction of threat. Many events cannot be totally prevented, but often the probability of them occurring can be reduced. This can include fire precautions, staff training, safety regulations, surveillance, etc.

Reduction of vulnerability. To reduce vulnerability is to lessen the risk of undesirable effects, given that some event has occurred. For example, in the event of an unauthorized person gaining access to a computer terminal, strong authentication procedures would reduce the vulnerability of the system as it would make it less likely that an intruder could cause damage to the system.

Reduction of impact. Even though precautions have been taken, events with undesirable effects can still occur. It is then important to reduce the impact of these effects. This can involve measures such as having an up to date backup when a hard disk crashes and having a spare disk handy to minimize down-time.

Detection. The sooner something unwanted happening in a system is detected, the better are the chances of successfully countering and reducing the effects. For instance, network traffic monitoring can often make it possible to fix problems before the situation deteriorates enough for the effects to be more generally noticeable.

Recovery. Should something disastrous come to pass, it is vital to be able to recover quickly. By planning ahead, an organization might be able to recover, for instance after a devastating fire, where an organization without contingency planning would not.

As can be seen, some security countermeasures are technical in nature, while others are not. Sometimes there is a tendency to overly trust technology to provide security, but as the examples show, technology can never provide comprehensive security. Upon closer examination it turns out that many wise precautions are purely administrative in nature, but must nevertheless be considered important security measures.

Achieving security is a perpetual process. The process starts by identifying threats and restarts when countermeasures have been established. A countermeasure should ideally eliminate a threat completely. However, this is often prohibitively expensive or simply not possible. It might be necessary to settle for reducing the threat, or for just reducing the severeness of potential consequences. Where there is a choice of countermeasures, a balance has to be struck between the cost of the countermeasure, the probability for the harmful event to occur, and the consequences of the event. For instance, very expensive countermeasures are often only deemed necessary when consequences are very serious. The striking of this balance is achieved when formulating the organizational security policies. This is an intricate activity as available resources next to always set a limit on what can be done.

2.2 Security and distributed systems

"Computer security" is a very broad area. One large reference work in the area, the *Computer Security Reference Book*, features 66 chapters in some

900 pages, but each chapter serves only to introduce a topic [JH92]. A reference work of this kind covers all aspects of the security process and takes into consideration many areas of application.

Most of what can be said about security in general also applies to computer security. In traditional security work, threats and risk are often physical in nature: theft, sabotage, fire, flooding, breakdowns, etc. These apply of course to computers as well. However, the area of computer security is special in that it also must deal with protecting *information* stored in a system.

What, then, can put information in our computer systems at risk? Without saying anything about what might, or might not, constitute harm to a specific organization, it is still possible to identify a number of categories of possible threats to an organization using computers. Threats to information turn into risks when an organization begins to become dependent on computer applications and information processed in computers.

To get a grip on these threats, we can start by looking at what is threatened. A coarse categorization results in three main groups: *threats to availability*, *threats to confidentiality*, and *threats to data integrity*.

Availability means that applications and information are available *when they are needed*. The more dependent an organization is on its information system, the more severe are the consequences of denied availability. Availability can be denied in many ways. Someone might steal a computer, the computer hall might burn down or get flooded, electricity might fail, networks can become over-congested, software might fail, file servers might go down, hard disks can crash, etc.

A threat to confidentiality is present whenever harm may come from computerized information being disclosed to an outside party. Here, harm might be to personal integrity, business, national security, or other items. What is defined as an outside party may also vary. It could be anyone but a single person, anyone outside a board of directors, anyone outside an organization, or even a specific set of individuals.

We touch finally upon threats to data integrity. Today many organizations rely heavily on computer information systems to provide data for making important decisions. Implicit here is a trust in that the information stored is correct. This is not, however, always the case. Again, depending on specific circumstances, the effects of using inconsistent or incorrect information may range from losing pennies to having space rockets take the wrong turn.

In addition to the direct consequences of threats of various kinds, there are in most cases also secondary consequences in the form of extra work and expenses to recover from the direct consequences.

We have already categorized threats according to the target of the threats.

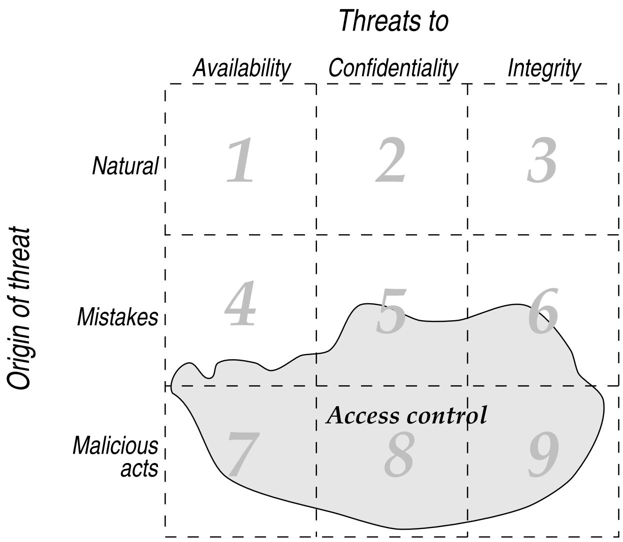

Figure 2.2: Threats categories where access control can help.

We can also look at the origin of a threat. We can, for instance, divide into one group those threats that come from nature and natural events, fire, floods, mechanical failures, etc., and into another group threats arising from the actions of individuals. This latter group can be further divided into threats where people cause harm inadvertently and where people act with the malicious intent to cause harm. If we combine this *"by whom"* categorization with the earlier one based on *"what"*, we get a matrix containing nine types of threats, as shown in figure 2.2. Given these nine categories we can examine countermeasures to see which parts of the matrix they cover. To achieve true security we would have to cover all categories completely. To exemplify, in the figure we have sketched an area that roughly corresponds to the protection offered by access control. Other areas require other measures. For instance, to cover area 1, threats to availability of natural origin, countermeasures could be to place computer installations on high ground, having smoke detectors, and fireproof walls.

2.2.1 Distribution

Today, most computer workplaces have some form of network connecting them to other computers, locally or globally. From at first having been mainly a way of pooling expensive resources, such as hard disks and printers, the communication services offered by the interconnecting network are more and more used to letting *applications* and *users* interact with each other. The fantastic growth explosion of the Internet shows there is an enormous potential for computer communication. The Internet explosion and the advent of the World Wide Web also show that existing technologies often are put to new uses, not originally anticipated.

In this section we look at distributed systems, and at security issues related to them. Before we can do this we need some conception about distributed systems.

When looking for examples of systems that are distributed we can immediately see that there are many kinds of systems that in one way or another can make claim to the title. Just to mention some, there are:

Distributed file systems. This is one of the most common applications of distributed systems. Not long ago, hard disks were expensive commodities and one way to see to that every megabyte was used as efficiently as possible was to use a centralized storage resource, accessed over a local area network. Most PC-based solutions are based on a client/server model where clients can incorporate a central hard disk drive as one or more additional "drive letters". Other, mostly Unix-based solutions, such as the *Apollo Domain file system* and the *Network File System*, are based on a peer-to-peer philosophy, where each machine in the network can be both a server and a client.

Distributed computing. Where many computers are interconnected in a network there is also a potential for dividing work among them. Analogous to how computing can be parallelized using multiple processors within the same machine, computing can also be distributed and parallelized utilizing multiple processors on multiple machines.

Distributed applications. Another form of distributed computing is where there is no central authority directing how processing is to take place. Instead, a number of autonomous systems, and users, independently perform parts of a larger task. Network communications are used to move results and administrative information among participating systems. *Groupware* software designed to help people collaborate in an office environment is an example of this class of applications.

Client/Server applications. Today, the most widely known example of client/server techniques in use is the World-Wide Web (WWW). Enormous quantities of information are requested by web-browser clients and delivered by web-servers every minute of the day. Often, web-servers themselves become clients when they query database servers to obtain data to be presented to their clients.

Taking these examples together, we can see that much of what is sometimes called the "IT revolution" could only have happened thanks to the advent of computer networks and various forms of distributed applications.

Security aspects of distribution

Networks and distribution have brought many advantages and new possibilities. Unfortunately, when we make our systems distributed, we also make them more vulnerable.

It is only natural that when we make use of new technologies, or old technologies in new ways, new aspects of old issues will emerge. To illustrate some security aspects of distribution, figure 2.3 elaborates on figure 2.2 by giving some examples of new threats introduced by distribution.

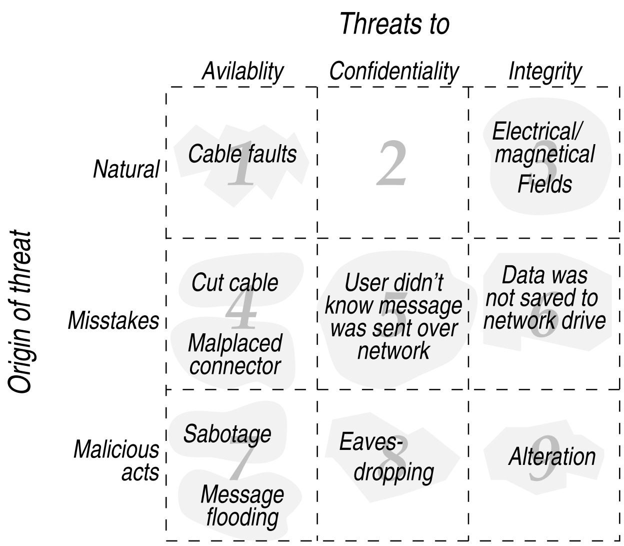

Figure 2.3: Threats introduced by distribution.

The figure shows an array of different possible threats. Natural events such as fire and flood may sever a communications link. A communications link subjected to electrical and magnetic fields might introduce erroneous bits in the data stream of a message, as can faulty contacts and other mechanical damage. A link can also be severed by mistake in many ways or one cable connector can be mistaken for another.

Distribution that is transparent might cause a user to inadvertently send sensitive data over an unsecured network, a transfer that would not happen if the user were aware of the risk. Another mistake would be for a user to store files locally instead of on a network drive, with the result that these files are not backed up.

An assailant may try to disrupt traffic among communicating parties in several ways. The actual link could be destroyed or the recipient can be flooded with messages making it impossible for legitimate messages to get through. A communications link gives ample opportunity for an intruder to eavesdrop on data sent over the link. This is especially true in public networks where it is relatively easy to write applications that listen for, and logs copies of, packets destined for some specific recipient or coming from some specific originator. An intruder might also insert himself between two communicating entities and attempt to alter information that is sent between the parties. He might also try to fake messages, pretending to be a legitimate originator.

2.3 Building blocks for secure systems

As was discussed in the previous section, it is apparent, and disconcerting, that distribution seems to open many possibilities for malicious attackers. In this section we introduce some building blocks that are important in order to counter these threats. The presentation here is kept simple and mainly introduces concepts and general principles. For a more in depth introduction to these topics we refer to chapter chapter 6 and to the literature. For instance [Pfl97] gives a good overview of the area.

The building blocks we introduce are *cryptography and confidentiality*, *authentication and non-repudiation*, and *access control*. Combining these building blocks into functioning systems is an important and sometimes difficult job. However, existing frameworks (see chapter 7) and plug-in libraries help make the task manageable.

2.3.1 Cryptography and confidentiality

Codes, ciphers, and secret writing has been used since ancient times to protect sensitive information from falling into wrong hands. *Cryptography* is about transforming a text, or arbitrary message, into something that is totally illegible, called *ciphertext* or *cryptogram*. It should also be possible to transform the *ciphertext* back into the original message, but only for the designated recipient.

In figure 2.4, Bob sends a message to Alice. To secure the transmission from eavesdropping, Bob first applies a cryptographic algorithm to the message. The algorithm is a function that takes two inputs, the message

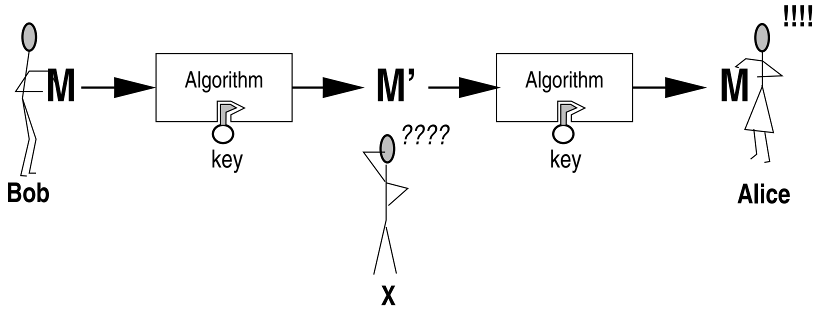

Figure 2.4: Encrypted communication.

and a *key*, and produces a ciphertext. Different keys produce different ciphertexts. To be able to read Bob's message, Alice runs the ciphertext through a cryptographic algorithm that is the inverse function of the one Bob used. The deciphering function also requires that Alice supplies the correct key. Distribution and management of the keys needed for encrypted communications are complex issues that are subject to current research.

2.3.2 Authentication and non-repudiation

An important issue for secure communications is to be able to positively identify the remote party. If this is not possible, an impostor might undetected take the place of a communicating party.

A process where one party proves its identity to another is called *authentication*. In most techniques for authentication, an entity proves its identity by demonstrating it *possesses* something that it is known to be possessed only by that entity. For instance, a common method for authentication of computer users is through passwords. This method works on the assumption that the password is only known to the system and the user, therefore, the system can conclude a user is who she claims to be if she can provide the password that matches her identity.

Another form of authentication is when a receiver wants to positively establish the identity of an originator of a message. For such purposes cryptographic techniques are used to generate a *digital signature* that is attached to the message. Digital signatures are described in section 6.1.4. A discussion about authentication in general can be found in section 6.2.

In many situations it is desirable for a sender of a message to obtain a receipt showing that a message was sent, and received intact, at a specific time. Examples are many, such as receipt of payment, proof that an application document has arrived on time, and proof that an addressee has received an email.

To make this possible, a common solution is to use a *trusted third party* that provides a *non-repudiation service*. A non-repudiation service uses cryptographic techniques to create an unforgeable receipt that can be uniquely associated with a message. When the service creates a receipt, it also inserts a time stamp that shows when the receipt was created.

If a recipient issues a receipt using a non-repudiation service, she cannot later deny having received a message. Another form of non-repudiation service is when a sender uses a non-repudiation service to obtain a receipt and a time-stamp showing when a message was created. If this receipt is attached to the message (together with a digital signature binding it to the originator), the originator cannot later deny having created the message at the time shown by the time-stamp. Applications of this can for instance be when signing contracts or writing a letter of debt.

2.3.3 Access control

During data transfer, confidentiality can be protected using encryption. However, when storing and managing information, encryption is often not used. Neither does encryption offer protection against all threats, for instance malicious erasure of data. Instead, sensitive information is locked away and an *access control system* guards who may and may not access the data. A good example of sensitive information is encryption keys used for software encryption. Another example is that it should be possible for a user to restrict the access by others to e-mail and other private material. Other concerns may be that expensive resources such as plotters and colour-printers might be misused, or that a software license limits the number of users who can use a program. Some systems store and process classified or secret information. In such systems measures must be taken to make sure only authorized users have access.

The generic name for a mechanism to achieve all this is *access control*. Figure 2.5 shows the principal components in an access control system. In standard terminology, the user or program that wants to make an access is called *initiator*. The program, file, or other resource that is to be accessed is called *target*. The access itself is called *operation*. To get a functioning access control system, all requests to perform operations must pass through the *access control enforcer*. For each request, the enforcer makes a decision to either allow or disallow the requested operation based on available access control information (ACI). Access control is described in greater detail in section 6.3.

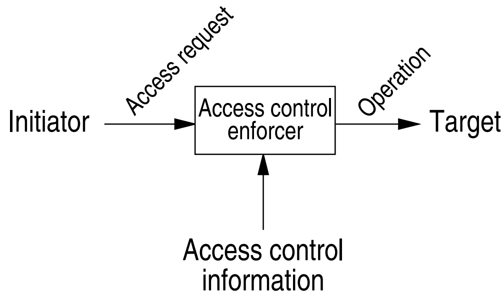

Figure 2.5: Main components in an access control system.

Chapter 3

A secure object-oriented architecture

In this chapter we present an architecture for a distributed secure system. More specifically, we sketch an object-oriented design based on a standard for distributed security services, ECMA-219. This design could be realized using a CORBA implementation, such as Orbix^{Tm} from Iona Technologies.

The basic design includes services for authentication and the issuing of *Privilege Attribute Certificates* (PACs). To get a comprehensive secure system, we also present an object-oriented adaptation of the *Generic Security Service Application Programming Interface*, GSS-API [Lin97]. GSS-API provides confidentiality and integrity services to communicating principals.

3.1 System framework components

Figure 3.1 shows the central components in the design. In later sections we look more closely at the various parts of the system. The basis for the working of the system is that there is a client machine where an application wishes to use services on a server machine. In order to be able to do this, the application must obtain a set of privilege attributes. Often, these are attributes granted to a human user who has been successfully authenticated by the security services.

GSS-API is a generic application programming interface to security services that is independent of the underlying mechanisms implementing them. By providing such an interface, it becomes possible to write applications that are portable between different systems and architectures at the source code level. GSS-API version 1 was introduced in [Lin93]. Recently, GSS-API version 2 was published [Lin97]. Version 2 retains and extends the functionality of version 1.

GSS-API *contexts* are an application-transparent mechanism for the exchange of security-related meta information between communicating parties. Actual client/server (or peer-to-peer) communications take place using ordinary communications channels, although when a GSS-API context has been established, data may be passed through GSS-API security enhancing services before being transmitted.

In order to make it possible for security services to control ongoing connections we mandate that (sensitive) client/server communications must be GSS-API enhanced. By representing GSS sessions as distributed objects, it is possible to include with the security services a context manager that can control these objects.

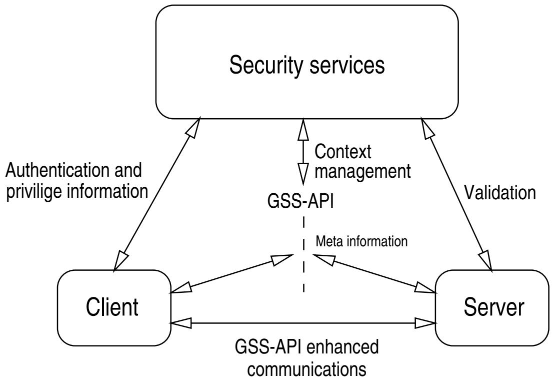

Figure 3.1: Abstract view of system.

3.2 Authentication and credentials

The first step, before any secure communications can take place, is for the principals involved to properly authenticate themselves. Once authenticated credentials needed for further interactions can be obtained. Credentials are used by a principal when initiating GSS-API-based services. However, GSS does not deal with, and is not intended to deal with, how a principal obtains credentials.

In our architecture, the basis for all security services is the framework specified in ECMA-219 [ECM96].

Figure 3.2 shows the components involved for establishing a set of credentials for a principal. Within the figure a number of *trusted objects* are indicated. The objects are trusted in the sense that we assume that a principal cannot disrupt their operation or otherwise access them other than through an interface devised for that purpose.

Furthermore, prudent security policies would mandate that the server objects implementing security services within the framework of ECMA-219 should be further protected by being placed on a machine that is physically secured, and that does not allow ordinary principals any access, except through the server interfaces.

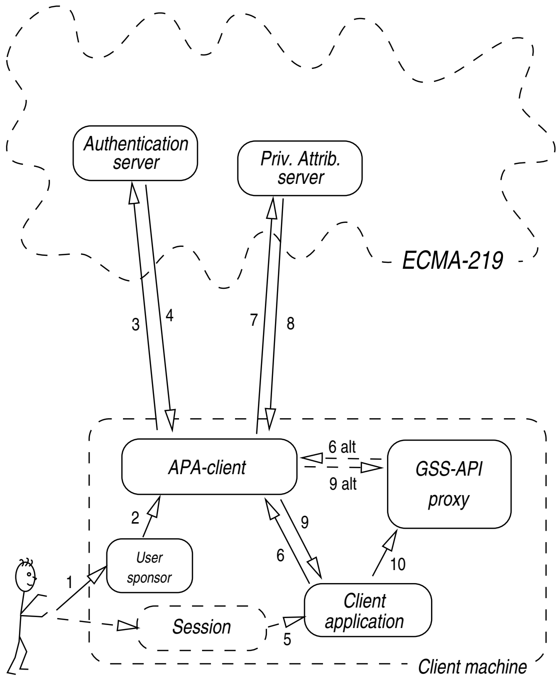

Figure 3.2: Entities involved in establishing a principal's credentials.

In the figure, a user wishes to be authenticated and then wants to invoke an application that will interact with a remote server. We will return to the object labeled GSS-API proxy in section 3.4. For now, it is sufficient to know that it is this object that provides an interface to GSS-API services to applications on the local machine. The exchanges and events illustrated by the figure are as follows:

1. A human user initiates operations by authenticating herself to the system. The object responsible for interactions with the user is called

user sponsor. The user sponsor determines what information is needed for proper identification according to the current security policy and requests this information from the user. In many cases this is a password known by the user. In more sensitive environments, the user might also be required to present a smart card or perform some kind of biometric identification.

The user sponsor is a trusted application that does not divulge or unnecessarily store passwords and other sensitive information.

2. The user sponsor serves as a front end to the ECMA-219-compliant services provided by the local APA-client (APA stands for *Authentication and Privilege Attributes*). The APA-client maintains per user security information and is responsible for initiating a *local session* as well. Through a local session, an authenticated user can interact with the system, using acquired credentials to invoke other services.

   The design of the user/local session interface is system dependent. If APA-services have been integrated with operating system security, a user perceives little difference from interacting with an ordinary workstation. In other implementations, the user typically interacts with a specially tailored shell program. This program makes sure that credentials obtained through authentication with APA-services are not directly available to someone who can impersonate the user's identity.

3. The APA-client uses the information supplied by the user through the user sponsor to contact the APA *authentication server*. This exchange takes place according to some established authentication protocol, such as Kerberos (see section 7.6).

4. Upon successful authentication, the authentication server sends an *Authentication Certificate*, AUC, back to the APA-client. The AUC contains proof of the principal's identity. The AUC is protected by applying cryptographic protection methods. This is done in such a way that the AUC is usable only for someone who possesses a key derived from the information supplied by the user sponsor. Thus, only the APA-client with which the user sponsor has communicated can successfully use the AUC.

5. Having obtained the AUC, the APA-client can initiate a *session* for the user.

6. When the user, through the session, invokes a *client application* that wants to establish secure communications, the application requests from the APA-client a set of privilege attributes. The client application needs credentials that allows it to access the GSS-API proxy.

7. The APA-client uses the AUC associated with the session that invoked the application to request from the APA *Privilege Attribute Server*, PAS, the necessary attributes.
8. Having verified the AUC, the PAS returns a *Privilege Attribute Certificate*, PAC, to the APA-client.
9. The PAC is passed on to the client application ...
10. ... that uses the PAC when it calls the GSS-API proxy to request that a *Security Context* is established between itself and the remote party.

As we can see, the key security-related objects on the client machine are:

The user sponsor that is the object acting as an intermediary between the user and the local security object.

The APA-client which is a collective name given to the local security object. The APA-client is responsible for interacting with central APA servers. The APA-client is also responsible for maintaining security-related information. In our design, the APA-client is responsible for instantiating session objects as well.

The sessions which are the objects that provide users with secure access to resources.

The GSS-API proxy which is the object that provides an interface that gives local applications access to GSS-API security services.

The services provided by the APA-client and the APA-servers are specified in ECMA-219. The standard also describes in more detail what information flows between these objects.

3.3 Access control

The mechanism used to implement access control is capability styled *Privilege Attribute Certificates*. A *capability* is a piece of access control information that is bound to an initiator and that the initiator presents to the access control enforcer. The capability itself contains all, or almost all, information necessary for the enforcer to decide if a requested access should be permitted or not.

To make access control role-based, the *Privilege Attribute Server* (PAS) that issues PACs must make use of a role-based security information database. As we here mainly want to demonstrate how an application perceives the system, we do not discuss the inner workings of the PAS. However, we envision the security information database to be built using the role framework presented in section 4.2.

3.4 Protecting network communications

Access control is a method to implement one class of security policies, and to protect against a certain class of threats. However, access control alone can never provide "security" in a broader sense. In this section we direct our attention to what goes on *between* a client and a server, when information passes over a network.

The overall goal is to achieve secure communications between a client and a server. In our model, the client and the server typically reside in different address spaces, on different machines. For this reason, communications must pass over the network and are thus vulnerable to any attacker who can get physical access to the network. Within an Intranet, it might be possible to have some control over the network. However, it is unlikely that it is possible to completely protect the network against a determined attacker. In the case of the Internet, it is clearly futile to expect to have any control whatsoever over what happens to a packet *en route*.

Thus, we need a framework for securing inter-machine communications. We assume here that intra-machine communications, i.e. communications between objects in the same address space, can be considered secure. This is a reasonable assumption as in order for such communications to be vulnerable an attacker must gain privileged access to the machine on which they take place. Should that happen, the measures we discuss here are of little value anyhow.

From what has been outlined above, transfer of security-related information ought to take place as inter-machine communications. However, at the same time we want to collect and centralize security-related functions within a security domain, even though clients and applications are distributed. (This is both for administrative and security reasons. A centralized security system is easier to maintain and can also be secured physically.)

In our object-oriented model, we achieve these goals by introducing, at each network node, a *security services proxy.* This proxy has the following properties

Intra-machine mechanisms are used to secure communications between the proxy and the entity utilizing security services.

The proxy shares a secret with the central *security and context manager* (SCM) that allows it to communicate securely with the SCM.

The proxy provides GSS-API services to applications on the machine where it is instantiated.

The proxy object is thus the link between client applications and the security services.

3.4.1 GSS-API

As a generic interface to security services, GSS-API defines two important concepts, namely *credentials* and *security contexts*, that are realized as data structures by GSS-API implementations. Applications only deal with handles to objects of these types and need not be concerned with the internals. However, applications must pass between them *GSS-tokens* that allow the context to be maintained at both communicating endpoints.

An application calls `GSS_Acquire_cred` to obtain a credential handle to a set of credentials for a specified name. An application need not concern itself with authentication and authorizations for the specified identity. Either this has been establish beforehand, or GSS will instigate an authentication procedure, or the call will fail. In many cases applications need not call `GSS_Acquire_cred` explicitly. Where there is no ambiguity, a set of *default credentials* can be used.

Credentials are used to establish a security context between two parties. An application initiates secure communications by using credentials obtained by default, or by calling `GSS_Acquire_cred`, to establish a *security context* with the remote party by calling `GSS_Init_sec_context`. A security context contains information about the logical link between two communicating parties. Important information includes, for instance, encryption methods to be used, and the shared conversation keys to be used for encryption. The GSS-API context also contains several state variables that describe the characteristics of the context. The arguments to the call are the name of the target and some parameters describing the preferred characteristics of the context to be established. If successful, the call returns a *context handle* and an *output token*. The application is responsible for passing the output token to the remote application, which in turn is responsible for forwarding the token to the local GSS-API implementation. Internally, a context contains information needed for secure communications, such as cryptographic keys.

When a context has been established, GSS-API offers per-message protection of communications. The basic services are integrity protection and confidentiality. Integrity protection of messages is provided by the functions `GSS_GetMIC` and `GSS_VerifyMIC` (MIC stands for *Message Integrity Code*, see section 6.1.4), while the confidentiality service functions are named `GSS_Wrap`/`GSS_Unwrap`. The GSS-API also supports replay detection and sequencing. If underlying mechanisms provide the necessary support, applications can receive alerts about duplicated or missing messages through GSS-API.

In addition to what has been described here, the GSS-API specification includes more functions for managing credentials and contexts. There are also a number of support functions for handling names and for getting information about the state of a context.

3.4.2 Security contexts

The GSS-API specification says very little about implementation. Through the token mechanism the specification provides a means for local GSS-implementations to pass among them the information necessary for establishing a context and performing other administrative tasks. Using tokens, it is possible to implement the GSS-API as completely autonomous local service providers.

Using the autonomous approach has advantages as well as disadvantages. Not having to rely on some central agent or service can increase flexibility and overall system robustness. On the other hand, not having any means for centrally monitoring system activities can be a security concern. For instance, without such monitoring it is very difficult to implement security policies such as *separation of duty* that dictates that a user must not act in two roles simultaneously.

In our object-oriented design, where objects transparently can invoke methods on objects on remote machines, we see an opportunity to introduce a mechanism for a coordinated management of security contexts.

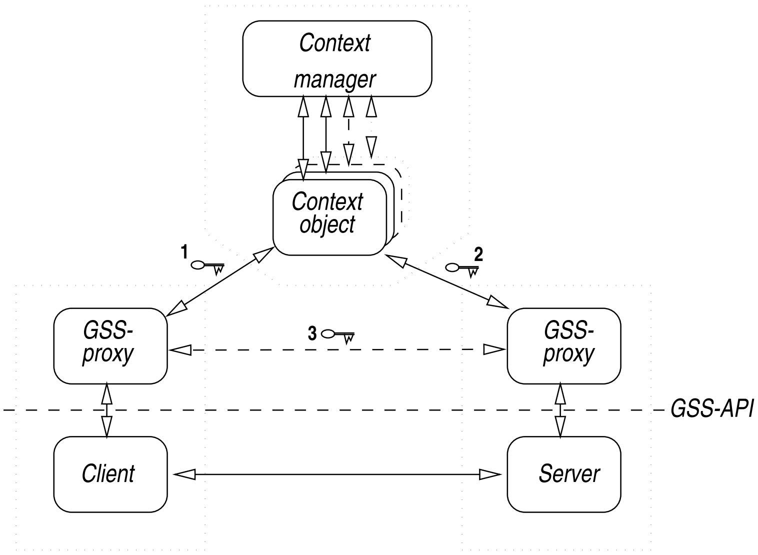

Figure 3.3: GSS-API context management.

Figure 3.3 shows the main components in our proposed GSS-API implementation. To implement the GSS interface, there is an APA collocated *Context Manager Object*. The context manager keeps track of created context objects and offers an administrative interface that allows a privileged system operator to examine and manipulate active contexts. Each context is represented by a context object instance. On all client machines there must exist an instance of the GSS-API proxy object. Applications interact with this proxy object only. This makes context management completely invisible to applications.

To protect communications among GSS-API proxies, the context manager, and context objects, keys are established per context among the context object and the proxies. These transient keys can be established using preinstalled symmetric keys, through asymmetric key cryptography, or through other key exchange protocols such as *Diffie-Hellman*. In the two former cases, it is necessary to install locally, on each machine, a secret key for use by the GSS-API proxy object. The Diffie-Hellman protocol has the advantage of not requiring a preinstalled secret in order for two parties to agree on a session key [DH76]. However, in the absence of preinstalled keys, authentication becomes difficult.

When an application requests a security context using its local GSS-API proxy to call `GSS_Init_sec_context`, the proxy forwards the request to the context manager. The context manager then creates an instance of a *context object* that is responsible for the actual management of the context. Context objects normally exist within the same address space as the context manager. `GSS_Init_sec_context` returns to the initiating application a token that contains a reference to the context object. The application can now establish a communications channel to the context target, and transfer the received token there.

Upon receiving a context token, the target application should relay the token to the local GSS-API proxy of the target. This is done by calling `GSS_Accept_sec_context`. The target GSS-API proxy can then call upon the context object to finalize the establishment of the context. When calling one of the context-establishing functions, an application indicates which characteristics it wants the context to have. The GSS-API implementation is not required to honor such requests from an application, but should do so if possible. Even though some requests are not fulfilled, the context will still be established. It is up to the application to examine the actual characteristics of a context and then decide whether or not the context should be used.

3.5 Legacy systems

Many organizations have large values invested in computer software. The task of integrating these legacy systems with new technology and system architectures can often be formidable. The object-oriented approach we present in this chapter offers a convenient short-cut for integrating legacy systems into a security framework. The idea is to turn an existing application into an object that can interact with other objects in the architecture.

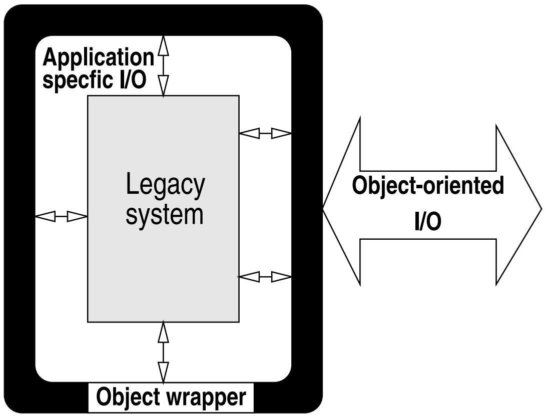

Figure 3.4: Encapsulation of legacy system

Figure 3.4 illustrates the principle for encapsulating a legacy system into an object. A wrapper software is written that outwardly has an object interface corresponding to the services offered by the legacy system. Internally, the wrapper object maps calls to the the object interface to the corresponding access to the legacy system. Output from the legacy system is likewise repackaged to fit the object-oriented paradigm. The main advantage of this approach is that the system being encapsulated needs not to be changed.

3.6 Conclusion

In this chapter we have presented an object-oriented architecture for a secure distributed system. Our high-level design shows how standardized or standard-based components can be used together to form a comprehensive system. The example also demonstrates to some extent the complexity inherent in the task of creating a secure and also usable distributed system architecture.

The object-oriented approach can be of great importance for existing applications. A common method for modernizing and adding new functionality to old and established applications, without the need for overly much modifying the base application itself, is to encapsulate the entire application into an object. The object interface can provide methods for all of the old functionality as well as for new additions. An application that has been encapsulated in this way can then be directly "plugged in" into the infrastructure architecture we have presented here.

Chapter 4

A role representation framework

Nowadays, software systems in many organizations are becoming highly integrated with, and part of, daily activities. This increased degree of integration raises demands on the ease of management and administration of software systems. At the same time as various systems more and more become integral parts of the activities of an organization, the organization becomes increasingly dependent on the systems. This increased dependency puts security requirements in focus.

Role-based access control, RBAC, is a paradigm that meets modern demands on integration and ease of administration for one aspect of security, access control. RBAC is also appealing as the main idea is intuitively simple and easy to grasp; permissions are given to roles, and roles are assigned to users as necessary.

In this chapter we discuss the concept of role and how this concept can be used for purposes of access control. Further, in chapter 5, we report on experience from how RBAC can be introduced into existing systems by creating an RBAC-aware NFS server.

4.1 The concept of role

Role-Based Access Control (RBAC) is an area that has received increasing attention in recent years. By separating the assignment of rights and permissions to roles from the assignment of users to roles, the security structure can be kept essentially static even though users may change over time.

The role concept, however, is wider than what is reflected in RBAC.

Further, we show how roles can be placed in a broader organizational perspective, and how roles viewed from this perspective relate to RBAC.

The concept of role has origins in the behavioral sciences and has long been used in enterprise modelling [Bid79]. Considering this background, Sloman defines a *manager role* as a set of authorization and obligation *policies* for which a particular *manager position* is the subject [Slo94]. A manager position is defined as a particular position within an organization[1]. Further, Sloman mentions four important facets associated with a position within an organization, namely *authority*, *responsibility*, *functions*, and *interactions*. Focusing on organizational policies, Sloman sees roles mainly as a tool for disseminating policies within an organization. This is also the approach taken by Marriott and by Lupu, Marriott, Sloman, and Yialelis [Mar93, LMSY95].

Sandhu, Coyne, Feinstein, and Youman adopt a similar view when they stress the importance of distinguishing roles as policy from the mechanisms used to implement them in an access-control system [SCFY94]. In the article, the authors derive a taxonomy of security models into which roles, and RBAC, are categorized as *computer policy models*. In [SCFY96] the same authors propose a family of reference models for RBAC. The base model is composed of *users*, *roles*, *permissions* and *sessions*. To this model is added assignments of *permissions to roles* and *users to roles*. In an extended model the authors also include *role hierarchies* and *constraints*.

A common operational approach, exemplified by Nyanchama and Osborn, is to define roles simply as named collections of privileges, whose specification may be subjected to any type of security policies [NO93a, NO93b]. Another approach is given by Demurjian, Hu, Ting, and Kleinman who propose a three-level, user-role definition hierarchy consisting of *user classes*, *user types*, and finally *user roles* [DHTK93]. A modified version of this model has been adopted by Mohammed and Dilts [MD94].

In the literature there is broad consensus about the administrative merits of RBAC. To us, one of the important benefits of RBAC is that it is possible to define a *static* security structure that is independent of individual users. By isolating the invariant parts, administration is reduced to a minimum. These advantages have led many authors to advocate RBAC in preference of traditional security approaches such as mandatory and discretionary access control (MAC and DAC) [CS92, DHTK93, SCFY94, SF94, MD94]. Sandhu and colleagues suggest that RBAC in a sense is *policy neutral*, as most other approaches, such as MAC and DAC, can be realized using RBAC [SCFY96].

[1] The word *manager* is here given a broad meaning denoting "someone who is responsible for certain operations".

Defining the role concept

As we have seen, there are many approaches to defining what constitutes a role. Not many authors place roles in a broader context or reflect on aspects beyond access control in their role definitions. We believe it is important to view RBAC, or any security approach, not only as an administrative tool or a purely technical issue of implementation, but also as one of several components in an overall effort towards good computer security. Even though technically advanced, any security scheme will lose its effectiveness if there do not exist well-defined *security goals*, and *security policies* derived from these goals, within the organization. It is also necessary that these policies are understood and endorsed by all those concerned. One goal in this chapter is to expand the role concept and place it in a wider perspective. In doing so we also want to make the distinction between the definition of roles and their uses clearer.

In the following we adopt the following, very basic, definition

> **(Role).** *A role is a position or job function within an organizational structure.*

From this definition it can be seen that, in order to completely describe a role, it is necessary to describe both the role *function* as well as its place in an overall structure.

4.2 Representing roles

Using roles as a descriptive tool for expressing policies and other organizational information can be very effective. In order to use such information in a computer system it must somehow be represented. In this section we present a flexible framework for a representation that has several important merits. Our framework provides a general way of modelling organizations by using roles and inter-role relations. Apart from being suitable for building models of organizations for use in information systems, we believe that the framework is a useful tool for analyzing organizations themselves as well.

4.2.1 Requirements for a role representation

To be able to put roles into a broader context, the representation of a role within an information system must allow for more than just privileges and access-rights. What are the requirements for such a representation? Apart from general requirements, such as *extensibility* and *modifiability*, commonly found in Software Engineering, we have identified the following general criteria

Formal foundation. In order to be able to make any predictions at all about whether or not a given role framework meets specified security goals and policies, it must be possible to reason about roles. This requires a formal notation as well as a set of rules and an inference machinery.

Generality. The model should be able to capture a wide range of facets of the role concept. We feel that roles are a powerful tool for describing organizations and that such descriptions have several uses besides that of access control. For example, Lupu and colleagues have made a strong coupling between organizational policies and roles [LMSY95].

Scalability. Any general framework runs the risk of becoming very complex in order to achieve completeness. To prevent role descriptions from becoming unwieldy, it must be possible to omit from an instantiation those parts of the framework that are not applicable to a particular case.

Applicability. In many cases, RBAC can be used to describe existing security policies. Role-based security schemes can also often be implemented, or emulated, using existing tools and mechanisms. It should be possible to apply a role description framework to these cases, too. By allowing for legacy systems in the framework in this way, comparisons between present and proposed systems become possible.

4.2.2 A role description framework

As mentioned when defining the role concept in section 4.1, there are two main components in any role description, the *role function* and the *relations* the role has to other roles. For roles to be meaningful, it must also be possible to assign users to roles. In addition to these components, it should be possible to associate *application data* with the role description. In the case of RBAC this may consist of privileges and permissions given to the role.

In this section, we present a framework that meets these requirements, as well as most of the general requirements identified in section 4.2.1. The complex task of providing the inference rules and machinery needed for automated reasoning about roles is subject to research.

The Role Descriptor Object

The basic entity in our role description framework is the *Role Descriptor Object* (RDO). The RDO is a representation of the external role concept used within the information system and contains the information associated with a role. We define the RDO to be 4-tuple <F, R, A, U> where

F	is a *function description* for the role
R	is an entity describing the *relations* the role has to other roles
A	is a block of *application data* associated with the role
U	is a list of *users* assigned to the role.

An *organizational model* $\mathcal{M}$ is a set of RDOs.

Function description

The size of the final organizational model depends greatly upon the granularity with which role function descriptions are made.

Taken to one extreme, it would be possible to allot a role to each *task* to be performed within an organization, resulting in a multitude of roles of type *performer-of-task-x.* This fits poorly with our given definition of a role, and would also be very cumbersome to administrate.

At the other extreme, we could try to make our role descriptions as general as possible in order to reduce the number of roles, and thereby also the administrative effort. Again, the resulting roles may fail to reflect the true organizational structure. For instance, although having only one *job*, many persons serve their organization in several different capacities.

Obviously, the exact granularity and extent of a role function description is highly dependent on the demands of the organization of which the description is a part. We believe that this judgment in practice will come naturally, provided the underlying organizational structure is well understood.

As it is hard to make any statements regarding the extent of the function description; it is also hard to enforce any specific structure to the description. At this stage we do not go further than to specify the function description attached to the RDO to be a textual one, intended for human readers.

Relations between roles

As the name suggests, positions within an organization are defined in terms of how they relate to other positions. A common relation of this kind is the *supervisor-subordinate* relation.

There may also be other relations between the roles in an organization. By using these relations it is possible to get other views of the organization. Lupu and associates mention several classes of such relations; *client-server*, *contractual*, *collaborative*, and *peer-to-peer* [LMSY95].

For each relation introduced into an organizational model, there must be a *relation specification* that describes properties of the relation such as name and type.

The designated *type* of any relation imposes constraints on how objects can be related. For example, ordering relations are anti-symmetric and generate graphs that are free from cycles. Symmetric relations, on the other hand, require that $(a, b) \in R$ **iff** $(b, a) \in R$.

There are many ways to structure an organization and also many ways to model these structures. It would be presumptuous to think that all conceivable structures and models can be expressed with a fixed set of relations. For this reason, relations should be definded on a case-by-case basis.

Formally, relations are often seen as a set of tuples $< o_1, o_2, \ldots, o_n >$ designating related objects (c.f. relational databases). In our framework, we do not specify any specific representation for relations.

Application data

In many applications this entity will be the largest. The application data entity of an RDO should be seen as a container for application information that may originate from many sources. Any application may specify an arbitrary object that can be attached to the application data entity of a role. In this way flexibility is preserved while the basic role framework is kept small. This also emphasizes the distinction between roles and the use of roles in various applications. In particular, access control information associated with the role is stored in the application data entity.

Assignment of users to roles

Regardless of how roles are otherwise defined or used, we must always be able to assign *users* to roles. Outside the information system this amounts to determining which *persons* should be able to take on a certain role. In the information system a *user* can be a person, but also a program (agent) acting on the behalf of a person. To represent a user we use a persistent *User Description Object* within the system.

In an RDO, the users that are capable of assuming the role are listed in a *Role User List*.

4.3 Role-Based Access Control

In the preceding we have presented a general, yet concise, framework for modelling roles within an organization. The framework can be used to describe various applications of the role concept.

Sandhu and colleagues have identified a number of important issues to be considered when designing a system to use role-based access control [SCFY94, SCFY96]. These issues include *Role definition and administration*, *Assignment of users to roles*, *Assignment of privileges*, *Role hierarchies*, *Constraints*, and *Role use*. In the following we shall examine how our framework can be applied to handle these issues.

In our examples, we will revisit the Acme General Hospital ward introduced in chapter 1.

4.3.1 Role definition and administration

Our framework implicitly defines at least three necessary logical information repositories. The *Role repository* provides other parts of the system with information about roles, their users and other information present in the RDO. The *Organizational relations database* contains information about the relations used to structure roles. In addition, there must also exist a *User database* that maintains unique user description objects for each user in the system. Which roles and relations that are to be defined depends on the organizational structure and policy.

In our example, the system designers at Acme General, after having carefully studied a typical ward in operation, arrived at a design consisting of seven roles and one pseudo-role (pseudo-roles are roles without a direct relation to any job function, see further section 4.3.4). Figure 4.1 shows these roles and also how they are related in an *is-a* hierarchy.

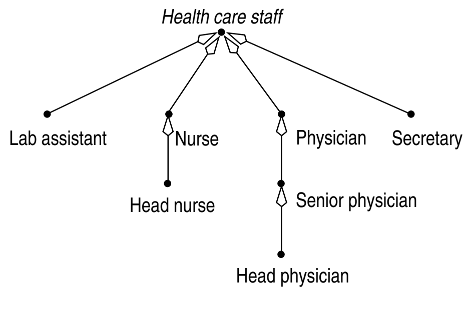

Figure 4.1: Roles at an Acme General hostpital ward.

It is often a complex task to analyze an organization in order to find the roles and relations that accurately model it. Using our framework to represent the model can help structure this work. In the case of RBAC, the explicit separation of the role definition from role use makes it natural to let the organizational structure govern the grouping of access privileges.

4.3.2 Assignment of users and privileges

An important motivation for RBAC is that the static security structure provided by roles simplifies the administration when users join, move within, or leave an organization. However, in large organizations the sheer number of roles and users may make centralized administration cumbersome. To alleviate this situation the authority to add or remove users to/from roles can be delegated in a controlled manner. In our Acme example the *head physician* is responsible for maintaining a duty roster that for each week assigns individuals to the various positions in the ward. It is also conceivable that the *head physician* further delegates some of this responsibility to the head nurse.

The task of assigning access privileges to roles can often be delegated as well. One should, however, keep in mind that this is a task separate from that of assigning users to roles. In many cases, these two tasks are performed by different roles.

Identifying which roles to use is a complex task, as is identifying what access privileges can be assigned to a role. A problem is often to find an appropriate granularity for the kind of operations that can be controlled in the access control system. If granularity is too detailed, manageability and overview are lost. On the other hand, if granularity is too coarse, it might not be possible to enforce all access control policies.

Given a set of roles and a set of operations, it is possible to create an access control structure. In our framework, only assignment of users to roles is expressed directly, assignment of privileges to roles must be realized through the application data entity in the role descriptor object. Abstractly, the privilege information is viewed as being stored *within* the RDO.

In figure 4.2, we show a simplified version of the access control assignments at our Acme General ward. There are several underlying policies for these assignments, the most important being those concerned with assuring the confidentiality and integrity of patient record data.

4.3.3 Constraints

Many security policies cannot be directly expressed in terms of access privileges to system resources. In RBAC, restricting rules or *constraints* can be

Role / Operation	*Health care staff*	Lab assistant	Secretary	Nurse	Head nurse	Physician	Senior physician	Head physician
Run office programs	X	X	X	X	X	X	X	X
Access word adm. info					X			X
Read patient record				X	X	X	X	X
Assign role								X
Append patient record			X			X	X	X
Sign patient record						X	X	X
Request test						X	X	X
Enter test result		X						

Figure 4.2: Hospital ward access assignments.

introduced at several levels. In order to avoid confusion it is important to identify the kind of constraints being discussed. We identify the following three main categories of constraints:

1. Constraints on the organizational model
2. Constraints on role activation
3. Constraints on role use

The first category can be referred to as *static constraints* while we call the other two *dynamic constraints.*

One important constraint policy is *separation of duty* that mandates that two roles must not have any common role users. An often cited example is that of a payment that must be authorized by two different roles, e.g. `clerk` and `supervisor`, to minimize the risk of fraud. Obviously, the purpose of this policy is nullified if a user can take on both roles. The separation of duty constraint policy can be implemented either statically or dynamically. In the static case, the constraint is placed on the organizational model, a user can only be assigned to one of two roles to be kept separated. In a dynamic implementation, a user can be *assigned* to both roles; however, for one and the same same user, the roles can only be active one at a time.

There are also several other types of constraints, as mentioned by Sandhu and colleagues [SCFY96]. These include *cardinality constraints* that can be used to limit the number of users that can be assigned to a role and *session constraints* that restrict the ways in which a user can activate roles.

In our framework, *inter-role relations* can be used to express static constraints. Mutually exclusive roles can be expressed as related through a symmetrical, anti-transitive, and anti-reflexive relation. This relation can be expressed as a list of pairs[2], e.g. $\{(r_1, r_2), (r_1, r_3), (r_2, r_4)\}$. The relation is anti-transitive, thus a user may take on both roles r_1 and r_4 (as (r_1, r_4) is not in the relation). At the same time, a user assigned to r_2 must neither be assigned to r_1 nor r_4. In some instances it may be required to generalize this binary relation to encompass three or more roles.

Simple cardinality constraints may be expressed using unary relations by defining one relation for each cardinality. This, however, is an awkward approach. There are also other constraints, such as *time limitations*, that are difficult to express as general relations. For constraints that do not involve other roles it therefore seems more appropriate to place such constraint information in the application data block.

Enforcing constraints

Apart from a method to describe constraints, there must also be means to enforce them. How this is achieved in detail is implementation specific. Still, a few general observations can be made.

Given an organizational model, static constraints can be seen as *assertions* about the model. A model in which all constraints are satisfied is called *consistent*. A way to preserve consistency is to check that changes to the model, such as the assignment of a user to a role, do not cause any violation of the constraints. The addition of new constraints is more difficult to handle. Imposing a new constraint may affect many roles in the model and have unforeseen ramifications. Therefore, there must also be a method for checking the consistency of an entire model.

To enforce dynamic constraints on role activation we introduce the concept of a *user session*. A user session provides a secure user interface to a system and is managed by a trusted *session manager*. When a user asks the session manager to activate a role, the session manager is responsible for checking applicable constraints before granting the access. Examples of such constraints are those that impose restrictions on which roles a user may have active simultaneously and constraints that put restrictions on the time of day a user may activate a role. One constraint is always checked; the user requesting access has to be listed as a role user.

Constraints on role use lie within the scope of operations of the *access decision function* in a system (see section 6.3.1). The decision function can

[2]The symmetry requirement implies that the pairs $\{(r_2, r_1), (r_3, r_1), (r_4, r_2)\}$ are also in the relation.

take into account various contextual information and decide to allow an active role to access a target only under certain circumstances (satisfying all constraints).

Categorizations

As we have seen, when looking at how constraints can be enforced, it is natural to make a distinction between constraints on the static properties of the organizational model and constraints on the actual use of roles. This categorization is only one of many possible. Figure 4.3 schematically illustrates some possible categorizations. One such categorization is that of *inter-role constraints* vs. *single role constraints*, e.g. mutually exclusive roles compared to cardinality constraints.

It is also possible to classify along the axis of *what is constrained* by the constraint. Mutually exclusive roles constrain the *assignment of users* to roles, whereas time limitations constrain the *use of a role*. Yet another category would be constraints on *privilege assignment* to roles.

Constrained entity	Number of roles involved: Two or more	Number of roles involved: Single role	
User assignment	X	X	Constraints on static structure
Privilege assignment	X	X	
User cardinality		X	
Time restrictions on role use		X	Constraints on system dynamics
Concurrency restrictions on role use	X		

"X" = Conceivable constraint

Figure 4.3: Different categorizations of constraints.

4.3.4 Role hierarchies

As we have seen, given a set of roles, we can within our framework introduce relations among roles to help structure the organizational model. Relations that introduce some kind of hierarchical structure among roles constitute an important class of relations. We can identify at least three important kinds of hierarchical relations:

1. Organizational seniority. This relation is the one commonly found in organizational charts. It shows for each role which other role within the organization is its immediate supervisor/manager/boss.

2. Specialization. In many areas, it is often natural for a role to be a specialization of another role. For instance, a *store-manager* could also be a *store-salesperson*, likewise being a *senior physician* implies being a *physician* as well.

3. Inheritance of access rights. When used for RBAC, access right assignments to roles often overlap, even when a role is not a specialization of another. Here the subset relation applied to access rights implicitly defines a partial order over the set of roles. By representing this partial order explicitly, the number of places where an access right needs to be specified can be reduced.

Even though all three kinds of relations can be found within an organization, there is very seldom a one-to-one mapping between them. For instance, a *company president* would be senior to everyone in an organization, however, the role would most probably not be a specialization of every role, e.g. the company president *is* not an *electricity technician*. Another example is *pseudo roles* (discussed by Parker and Sundt in [PS95]). When using roles to group access rights, it is often natural to introduce a role with the sole purpose of being a container for access rights common to a number of other roles. A pseudo role of this kind does not correspond directly to any role in the "real" organization.

In the literature, the different kinds of hierarchical relations are not always kept separate. For instance, Nyanchama and Osborn have proposed a model where roles are organized into a lattice. The lattice is based on a "*junior-senior*" relationship that implicitly suggests an organizational seniority relationship. The lattice is defined purely using set inclusion of access rights [NO94].

Within our framework, it is possible to represent all of the different kinds of hierarchies. However, we do not believe that organizational seniority relationships should be used for inheritance of access rights, as doing so can lead to an undesirable concentration of rights at the management level. On the other hand, to identify and chart this relation is important in order to understand an organization. Such understanding is essential in the policy making phase of the security process.

Once security policies have been established, specialization and explicit inheritance relations can be used to create a structure within an organizational model that is efficient for access control purposes. Keeping the distinctions we have made in mind, inheritance relations among roles can help reduce the number of explicit privilege assignments and can also provide a better overview of access control information.

We discuss here the relations for inheritance of access rights among roles together with a definition of the set of effective rights for a role. We start by making some definitions:

Let $\mathcal{R}$ be the set of roles in our model

$\mathcal{P}$ be the set of all privileges that can be assigned to a role

$I : \mathcal{R} \times \mathcal{R}$ be the relation *inherits-from* used to express explicit inheritance of access rights

$I_{Tr} : \mathcal{R} \times \mathcal{R}$ be the relation *includes-role* used to express transitive inheritance of access rights

$\Psi : \mathcal{R} \rightarrow \mathbf{2}^{\mathcal{P}}$ be a function returning the set of privileges *assigned* to a role

$\Upsilon : \mathcal{R} \rightarrow \mathbf{2}^{\mathcal{P}}$ be a function returning the set of privileges *inherited* by a role.

For brevity, where a role r is implicit, we will write Ψ and Υ to denote $\Psi(r)$ and $\Upsilon(r)$ respectively.

The function Υ is defined in the following manner:

$$\Upsilon(a) = E \cup X$$

where

$$E = \bigcup_{(a,x) \in I} \Psi(x)$$

and

$$X = \bigcup_{(a,x) \in I_{Tr}} \Psi(x)$$

The set P_e of effective rights for a role is defined as $\Psi \cup \Upsilon$.

In prose terms, this means that if role **a** *inherits-from* role **b** ($(a, b) \in I$), then the privileges assigned to role **b** are inherited by role **a**. This process is repeated for each role **a** *inherits-from.* However, any privileges that role **b** might have inherited are not transfered to role **a**. In addition, rights can also be inherited through the transitive relation I_{Tr}.

Figure 4.4 (a) shows a simple example where privileges are inherited using the *inherits-from* relation.

If some roles are specializations of other roles, for instance *electricity-technician — technician* and *company president — manager*, we can take

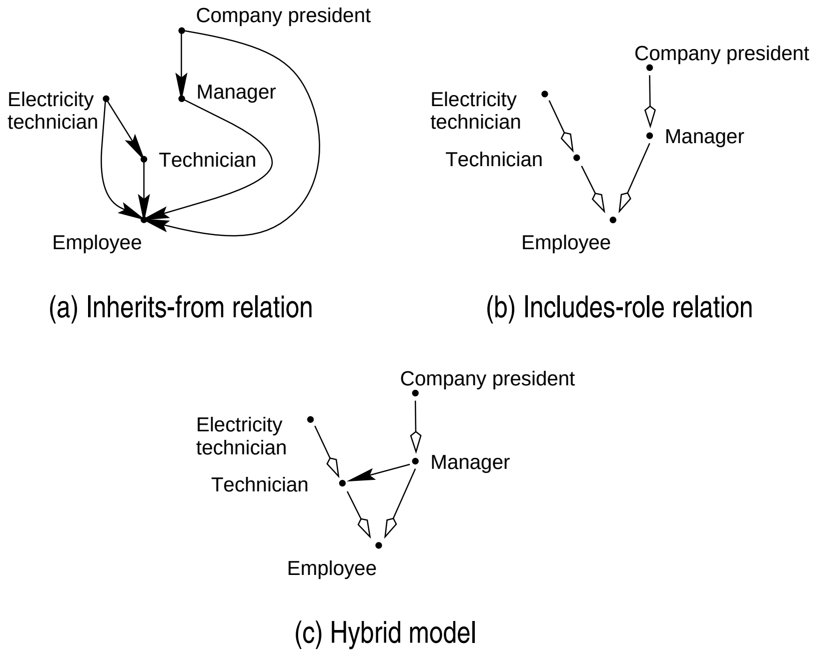

Figure 4.4: Privilege inheritance.

advantage of this structure and avoid having to repeat the link to the pseudo role *employee*. We do this by using the transitive inheritance relation, *includes-role*, denoted I_{Tr}.

Figure 4.4 (b) shows the same rights being inherited as in (a), but expressed using I_{Tr}.

The relation I_{Tr} can be expressed in terms of I. However, use of I_{Tr} can reduce the number of explicit links in the model. The two notations can be combined, as in figure 4.4 (c). Here, the role *manager* is specified to inherit the privileges assigned to the role *technician*.

We believe that role hierarchies for the purpose of access control should primarily be seen as administrative tools that are not visible to the user. By defining role hierarchies it is possible to avoid having to duplicate the assignment of access rights to similar roles.

4.3.5 Role use

In section 4.3.3 we introduced the concepts of *user session* and *session manager*. The session manager handles the dynamic side of RBAC that arises when users take on and relinquish roles. We have not yet included these entities into our framework in a formal way.

The primary task for the session manager is to grant access to roles and to convey to the user the privileges associated with these roles. As we have mentioned, granting role access involves checking a number of constraints, the basic one being that the user should be listed in the list of allowed users for the role.

The nature of the privileges granted to a user varies greatly among implementations and organizations. For instance, there is a large difference between a centralized and a distributed system. A centralized system can implement an access server that mediates all access to protected system objects, whereas in a distributed system privilege information has to be conveyed to remote applications.

4.3.6 Implementing the framework

We have observed that to maintain an organizational model created within the framework, a number of databases are necessary. In order to maintain the consistency of the model it is necessary to mandate that access to these repositories must be through an API of primitives that operate on the model. These primitives can then be designed to guarantee to maintain consistency.

There are a number of issues to be addressed in this design. For instance, as individuals in an organization in many cases have several responsibilities, it is often desirable that a user is able to acquire, and act in, the capacity of several roles. This raises the question of how a user should go about acquiring and relinquishing roles. To acquire a new role, the user can make a request to the system using her previously authenticated identity. Alternatively, the user can renew authentication, specifying the desired role.

Another issue is how a user should go about relinquishing a role. This can take place voluntarily, but security policies may *require* a role to be relinquished at some point, regardless of the principal's wishes in that respect. An example is when there is a policy of *separation of duty*. Here, the policy *mandates* that a principal cannot act in role **a** at the same time as it also can act in another role, **b**. If roles **a** and **b** both are assigned to a principal, the system *must assure* that role **a** is relinquished whenever role **b** is acquired, and vice versa. Separation of duty may also be enforced through a policy stating that a principal must not be assigned to both roles. This, however, is a static constraint on role database contents, not on role usage.

To implement mechanisms that guarantee that a role is relinquished when mandated can be difficult. This is because in distributed systems, information about activated roles is often passed to a principal in the form of an autonomous Privilege Attribute Certificate, PAC (see section 3.2 and 3.3.) After a PAC has been issued, the security attribute server no longer has access to it. To work around this problem, the PAS can maintain a list of PACs that have been invalidated. In order to enforce policies such as separation of duty, servers would then always have to check whether a presented PAC is still valid, something that could introduce substantial overhead in the system.

Another approach is to introduce an entity such as the context manager proposed in the architecture of chapter 3.

Examples of policies mandating privileges to be relinquished at a specific time are, for instance, policies that restrict the times when a role can be activated to certain time periods. Such limitations are relatively easy to enforce by limiting the time of validity of a PAC to conform with the policy.

4.4 Related work

Within the frame of our project, Karin Rappe has presented a model and an implementation based on a refinement of the framework presented here [Rap97]. She implements both explicit ("limited") inheritance, as well as transitive ("full") inheritance among roles. In her work, Rappe has extended the role description entity to include an *authorization function* that is called at role activation time to check dynamic constraints on role usage.

4.5 Conclusion

The framework presented in this chapter is a general one. The framework is useful in that it distinguishes between various aspects and uses of the role concept. The framework provides an abstraction whereby it is possible to model an organization through the roles that can be identified within in it, and the interrelationships these roles have. *After* this model is created, it can be put to different uses, such as access control.

Chapter 5

Implementing RBAC using NFS

Whenever new technologies or novel designs and ideas emerge in some area, there arises a problem of how to introduce the new without having to entirely discard the old. In an area as rapidly evolving as computer science, this problem is accentuated. Design compromises must often be made in order to accommodate what are often called *legacy systems*, into which much money and competence are invested. In chapter 3, we pointed at the possibility of integrating old systems into new environments by encapsulating them into autonomous objects. In this chapter we look at a less general but more concrete approach where we propose small changes to an existing distributed infrastructure to make it possible to introduce RBAC into the system.

As we have argued in chapter 4, RBAC provides a conceptually simple model for organizing and representing access control information and can provide facilitated administration and better overview of security information.

However great the benefits of adapting a role-based view for expressing access control information may be, the issue of legacy systems remains. In this chapter we present a design that enables the introduction of RBAC into a distributed system that uses the Network File System (NFS). A prototype implementation shows that our design can be realized with little effort and with acceptable impact on general operation and performance.

5.1 The Network File System

The Network File System, NFS, was introduced by Sun Microsystems Inc. in 1985. The goal was to provide transparent access to remote file systems. Sun chose to make NFS an "open" protocol by publishing the specification and by licensing a reference installation. This has made NFS widely used and ported, to non-Unix platforms as well. The first published version of NFS was called Version 2 [Sun89]. In 1995, several vendors, including Sun, IBM, and Digital, published an extended Version 3 of the NFS protocol [PJS+94, CPS95].

NFS is a client/server protocol. An NFS server makes file systems available to clients. When attached to a server, a client application will perceive the file system exported by the server as being a part of its own local file system.

Figure 5.1 shows the principal components used in NFS client/server communications. Important pre-requisites for NFS are the *Virtual File System* (VFS) and the *Remote Procedure Call* mechanism (RPC). The VFS is an interface to an abstract file system and provides the indirection necessary in order to present a single file system to the client while physically there may actually be several. RPC, together with the *External Data Representation* standard (XDR), provides the network transport mechanism used by NFS. The VFS, RPC and XDR are all technologies introduced by Sun [Kle86, Sun87, Sun88].

The operation where a client incorporates a remote file system into its own virtual file system is called *mount*. This initial exchange is specified in [Sun89] as a companion *mount protocol* to the NFS protocol. During the mount exchange the client receives an initial NFS *file handle* from the NFS server. The client passes a handle of this kind on every subsequent call to the NFS server. A file handle contains all information necessary for the server to carry out a request. This feature of the NFS protocol makes the server *stateless* as the server need not maintain any information about connected clients. This is a very important property of NFS as it makes the protocol very robust. For instance, there is no need for any special procedures to handle server crash recovery. If the server goes down, a client will simply retry its request until the server again comes online to serve it.

5.1.1 NFS service functions

Once a client has mounted an NFS file system, the NFS protocol specifies a number of functions a client may use to access the server. In NFS version 2 there are 18 functions (of which one is obsolete and one is for use in later versions), whereas in version 3 the number has been extended to 22. As our

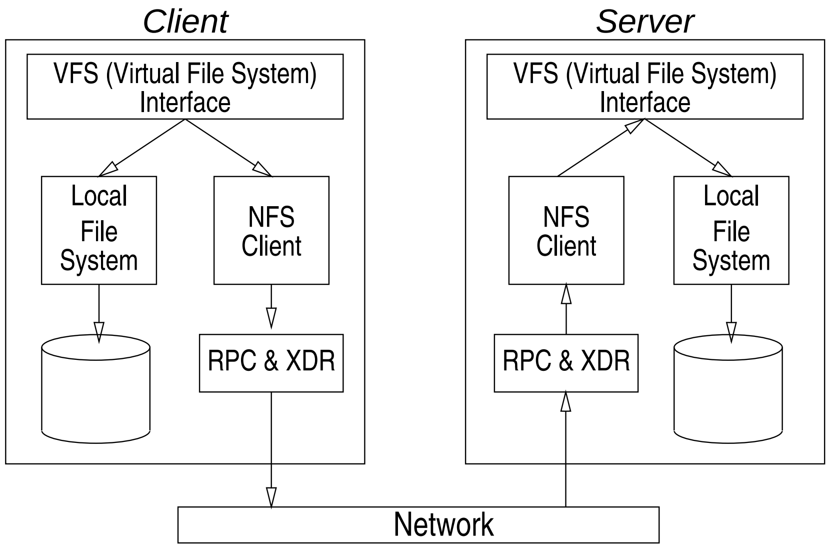

Figure 5.1: NFS communications between two Unix computers

design is based on a version 2 server[1] we discuss here only functions found in that version.

	Name	Description
00	NULL	Do nothing
01	GETATTR	Return file attributes
02	SETATTR	Set file attributes
03	STATFS	Return file system status
04	ROOT	**Obsolete** (Get file system root)
05	LOOKUP	Lookup named file and return file attributes and a NFS file handle
06	READLINK	Read from a symbolic link
07	READ	Read from a file
08	WRITECACHE	(To be used in later versions)
09	WRITE	Write to a file
10	CREATE	Create a file
11	REMOVE	Remove a file
12	RENAME	Rename a file
13	LINK	Create a hard link to a file
14	SYMLINK	Create a symbolic link to a file
15	MKDIR	Create a directory
16	RMDIR	Remove a directory
17	READDIR	Read the contents of a directory

Table 5.1: NFS version 2 requests

Table 5.1 summarizes the service functions defined in NFS version 2. Upon examination it is evident that the NFS services rather closely mirror file level operations available in a Unix system. However, by using a subset of the functions, or by providing suitable mappings, NFS has also been implemented for other operating systems, such as MS-DOS, Novell Netware and VMS (see for instance [Mic, Nov, Att]).

5.2 Design for the RBAC NFS server

Before we can make an access control decision, the identity of the principal requesting the operation must be established. In a secure system, strong authentication should be used. More specifically, many implementations of

[1] The base server we use does not yet support NFS version 3.

RPC (upon which NFS is built) can be run "secure", meaning that RPC datagrams are strongly authenticated. However, the NFS specification ([Sun89]) does not define the permission checking to be used by servers.

In our demonstration implementation, we have not addressed the issue of strong authentication, instead we use the AUTH_UNIX style authentication that is used by default in most RPC implementations. In the future our implementation can easily be modified to make use of other forms of authentication. Another issue that we have not addressed in this demonstration is the fact that in spite of proper authentication and access control mechanisms, an intruder might still be able to obtain access to communications that pass over a network. In the case of NFS, data transferred over the network is not normally encrypted, so an eavesdropping intruder may read any file read or written using NFS just by monitoring the traffic. An active intruder might also attempt to modify the information passing through the network. To guard against these threats, additional measures, such as IP-level encryption (see section 7.1), are necessary.

Using AUTH_UNIX style of authentication, the server receives with each request the effective user and group identities (UID and GID respectively) of the caller. During normal operations, these identities are used when the server makes file accesses. However, the main responsibility for access control still rests with the kernel of the client that allows or disallows access based on the file attributes of a file, as reported by the NFS server.

Figure 5.2 shows the main components of our NFS server implementing role-based access control. The central idea of our design is to mount the remote file system as usual but then to side-step the default permission checking behavior. The numbers in the figure show a typical flow of events.

1. A user starts out by using a special application to *activate* one or more roles. This application communicates with the **role state** monitor module implemented together with the NFS server itself. A user can only activate roles to which she has been assigned in the **User/Role database**. Information about active roles is kept in the **role state** module. As a rule, a user must independently establish an active role before she can access files through the RBAC NFS server. The exception to this rule is when access to some directory or file has been explicitly granted to a user, for instance to a home directory.

2. After having activated a set of roles, a user application can access a file on the remotely mounted file system. When this happens, the client machine sends an NFS service request to the server.

3. When the server receives a request, it uses the UID of the calling client process as a key to acquire information about what *roles* the user has

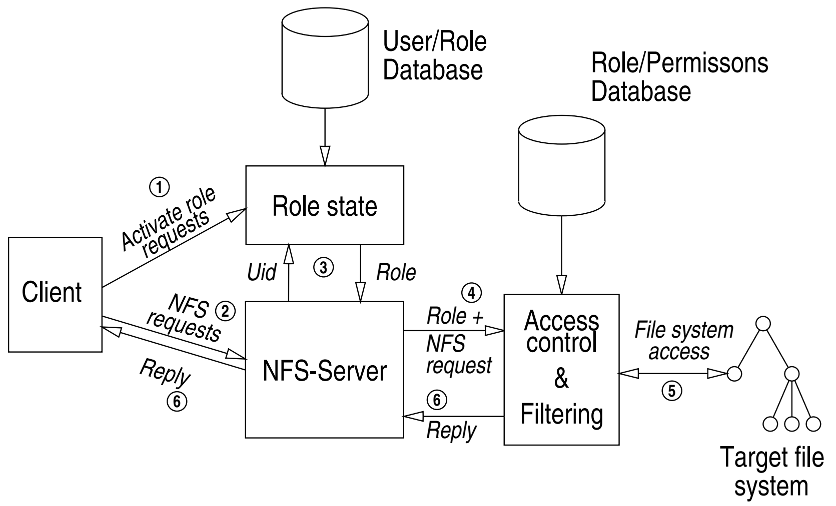

Figure 5.2: Main components of the RBAC NFS server

active.

4. Having retrieved the currently active roles for the user making a request, the NFS server makes an access control decision for the request using the information in the **Role/Permission database**. This database assigns permissions to roles authorizing them to access files, directories or directory trees.

5. If the requested access is permitted, the corresponding operation is carried out on the target file system.

6. In our RBAC NFS server design, the owner and permission attributes stored for a file on the **Target file system** are not used. Instead, in a **filtering** step, those NFS requests that query permission attributes are intercepted and attributed values based on the information in the **Role/Permissions Database** are substituted in the reply. The result is then passed back to the client.

Our design implies a change in the conceptual model describing the remote file system. User identity (UID) and group identity (GID) are no longer central concepts, except for conveying information that can be used to tie the UID to a role and for providing access to files that are indeed personal in nature.

However, an important design goal is to achieve a solution that requires no, or minor, changes in client systems and applications. In order for the design to be useful, it must be possible to run most existing applications without modifications.

Since client programs should run unmodified, it is obvious that whatever access rules and access modes we define using the **Roles/Permissions Database**, we must ultimately map these permissions into file attributes used by the client.

5.2.1 Unix permissions and access modes

In the basic Unix file system model, each file or directory has two important attributes, *owner* and *group*. Access rights can be specified to apply for the owner, users belonging to the same group as the group attribute of the file, and for all users not in any of the other two categories. These categories are commonly referred to as *"user"*, *"'group"*, and *"others"* respectively. There are three access modes, *read*, *write*, and *execute*, abbreviated `r`, `w`, and `x` respectively. When listing access attributes, these letters are used if the corresponding access has been assigned, if not, a hyphen, '-', is used instead. In verbose directory listings, an initial character position gives information on the type of file, followed by three groups of access specifications, for owner, group and others respectively. For instance, '`-rwxr-x---`' indicates a directory file to which the user (owner) has *read*, *write*, and *execute* permissions, members of the group the file belongs to have read and execute permissions, and where other users have no rights at all.

Although there are only three kinds of permissions, we can identify ten basic kinds of access modes, *create/delete file*, *create/remove directory*, *read/write file*, *execute file*, *search directory* and *create hard/symbolic link*.

Table 5.2 shows the necessary permissions in order to access a file or directory in different ways. In addition to the permissions shown in the table, it is necessary to have at least *execute* permission to all directories in the path leading to the accessed file.

The kernel of the client machine enforces access control using the permissions assigned to a file. This is also the case when the file system is mounted over NFS, it is still the client machine that is the access control enforcer, using the permission attributes reported by the NFS server.

5.2.2 Permissions and access modes for our design

The granularity of Unix permissions and access modes is limited.

From table 5.2 it can be seen that there are some access modes that only can be granted together with others. For instance, it is not possible to

Access mode	Necessary permissions	
	To file	To parent dir
Create file		`-wx`
Create dir		`-wx`
Remove file	`-w-`	`-wx`
Delete dir	`-w-`	`-wx`
Read file	`r--`	`--x`
Write file	`-w-`	`--x`
Execute file	`--x`	`r-x`
Search dir		`r-x`
Make symbolic link		`-wx`
Make hard link		`-wx`

Table 5.2: Correspondence between access modes and necessary UNIX permissions

allow file creation without at the same time allowing creation of directories.

An access mode that often is desirable is *append file*. However, this mode is not present in the basic Unix model. There is also no way to express that a permission should apply to an entire directory tree.

In the design of the RBAC NFS server, we have tried to address these shortcomings by introducing a different set of permissions. In our design, the basic permissions *read*, *write*, and *execute*, remain the same. However, we make it possible to distinguish between access to files and access to directories. We also introduce an *append file* permission and allow permissions to apply generally to a directory subtree.

An important file access mode is the operation of changing the file permissions. This is important in a Unix system as it is the characteristic mark of a *Discretionary Access Control* (DAC) system. In a DAC system, the owner of a file or directory may change the permissions for that object [DOD85]. However, RBAC is not inherently discretionary in nature. In our system, we use a permission database that assigns file access permissions to roles in an independent manner. Access to modify this database is not given out by default. Applied to our design, this could lead to difficulties. For instance, it would not be possible for a user to write a usable shell script if she could not set the executable permission for the file. To address this problem, we include a special permission that allows this specific operation.

Given these considerations, we decided to implement explicitly permissions for eleven different access modes. These permissions are summarized in table 5.3. For each permission, the table lists a name and a mnemonic that is used when specifying permissions.

Operation	Mnemonic
Read file	FR
Create file	FC
Write file	FW
Append to file	FA
Delete file	FD
Execute file	FX
Create directory	DC
List directory	DL
Remove directory	DR
Toggle execute bit	XT
Create symbolic link	LC

Table 5.3: Permissions in the RBAC NFS server

5.2.3 Special cases

Some special issues were considered when designing the RBAC NFS server. None, however, constitutes any serious problem and we therefore discuss them here only briefly. The special cases we identified were:

Symbolic links. No special steps need be taken in order to properly handle symbolic links. The permission attributes of a symbolic link are never used. Instead access control is performed when the target of the link is referenced. By convention, the permissions for a symbolic link are listed as "`lrwxrwxrwx`". We, too, adhere to this convention when returning attributes for symbolic links.

Hard links. Hard links are in fact an alias mechanism that allows a file to have more than one name. This can often lead to confusion. For this reason we do not allow hard links to be created, although existing links in the server file system work correctly, as hard links by definition are indistinguishable from the file itself. We do not believe that this is a significant limitation as symbolic links can be used instead. Also, in our experience, hard links are seldom used in practice. Should the need arise, our server can be extended to handle hard links as well.

Device files. Device files are always created directly on the server file system using the `mknod` command, never using NFS. Indeed, there is no NFS mechanism for creating special files. The handling of access to device files is no different from that of access to other files.

Set UID/GID attribute. Unix file systems offer a special *set user/group id upon execution* attribute for executable files. This is an important tool for security administration. As our server does not take into account owner or group ID of a file, the concept of *set uid/gid* becomes undefined. Therefore, at present, we do not support this attribute. This amounts to assuming that all NFS file systems are mounted using the `nosuid` option.

5.2.4 Presenting file information

As user and group access attributes are not used, a natural approach is to only use the *"others"* group of file access attributes to convey access control information from the server to the client. This is also the default behavior in our system. As mentioned, one exception to this rule is made for symbolic links, by convention these always have the permission set to "`lrwxrwxrwx`". In order for the *others* attributes to apply, it is necessary that none of the *user* or *group* attributes apply. We achieve this by presenting all files and directories as belonging to user `nobody`, group `nobody` (having UID/GID 65534).

However, upon closer examination, it turns out that toggling of the *execute* bit cannot be implemented using this presentation layout. As Unix only allows the owner of file to change the file attributes, a request to do so for a file to which the user is not the owner does not succeed. Obviously, this would be the case if the owner is set to *nobody*. The request would be blocked by the kernel of the client machine and not even forwarded to the NFS server. To amend this situation, we set the owner of a file to be the current user if the effective permissions to the file include the *execute bit toggle* permission (XT). In this case we must also use the *user* position when specifying access attributes. This solution also has the positive side effect of making a visible distinction between files to which XT permission is granted and files to which it is not. Figure 5.3 shows a typical directory listing. In the listing, read access is granted to the files `FILES`, `README` and `RELNOTES` through role membership. `INSTALL` and `linux` are links. Effective access rights for these entries depend on what rights are granted to the target of the links. `test.txt` is a file to which the user is granted read and write access through role membership. Access rights to the directory `mail` are through explicit assignment. Finally, the entry for `fixit` is an example of a file for which the user `jsmith` can toggle the execute permission.

5.2.5 A syntax for specifying RBAC permissions

Permissions are assigned to roles in order to express an *access policy* for some system resource. In the case of the RBAC NFS server, the system resources in question are directories, directory trees, and files[2].

For simplicity, the current implementation uses a text file for specifying permissions to access files and directories assigned to roles. Some of the details in this section are specific to the representation we currently use.

The syntax for writing lines in the permission specification file is shown in figure 5.4. In the syntax, *path* can either denote a directory or a file. Permissions specified for a directory are assumed to propagate recursively down the directory tree rooted in that directory. When determining which permissions apply to a given directory or file, the entry with the *path* constituting the longest prefix of the accessed file is always used. This makes it possible to override permissions at points further down a directory hierarchy.

The special syntax " **string** '`/*`' " is used to specify a permission policy to be applied to all *sub-trees* of the directory specified by **string**. However, the policy does not apply to the directory itself.

Another special syntax is the pseudo-role **everyone** that can be used to introduce a policy that always applies, regardless of which roles the user may have active.

Permissions are expressed using a sequence of colon-separated mnemonics, for instance `FR:FC:FW`. There is also a shorthand notation for this where `FR:FC:FW` is written `F=RCW`.

The algorithm for establishing the effective permissions for a given *target-path* is to start out with an empty set of *effective permissions*, **P**. We then look in the database for the entries having the longest prefix of *target-path* in their path component. For each such entry we check to see if it has a permission specification for any of the active roles, and for the pseudo role **everyone**. Any permissions thus found are added to the set **P**.

Figure 5.5 shows an example permissions database. This database describes what access various roles have to different parts of the *server* file system. In the example, the role **sysadm** is given complete permissions to the entire file system. By assigning the *directory list* (DL) privilege to '/' for the pseudo role `*everyone*`, all users are allowed to browse the contents of the file system. However, when determining the access to a path for a given role, the rule having the longest matching prefix is used. This means that it is possible to block access for `*everyone*` to the directories `/sbin` and `/usr/sbin` by stating alternative rules for these directories, that apply to `*everyone*` but convey no permissions. The line for `/usr/apps/dbms/audit.log` demonstrates that access can be specified, and differentiated, for specific files. The next line, for `/usr/apps/dbms`, gives *file read* and *file execute* rights for the directory to the roles **manager** and **clerk**. The last line demonstrates how it is still possible to assign a

[2]In Unix, the principle is to treat *all* devices as files. The resource concept thus encompasses more than just sequences of bytes stored on disk.

```
-------r--  1 nobody  nobody  259709 Feb 27 20:15 FILES
lrwxrwxrwx  1 nobody  nobody       8 Feb 11 16:00 INSTALL -> /var/adm/
-------r--  1 nobody  nobody   27954 Feb 27 20:14 README
-------r--  1 nobody  nobody   24541 Feb 27 20:16 RELNOTES
lrwxrwxrwx  1 nobody  nobody      14 Feb 11 16:00 linux -> /usr/src/linux/
drwx------  1 jsmith  nobody    1024 Feb 11 16:45 mail/
-------rw-  1 nobody  nobody    2265 Feb 15 17:17 test.txt
-rwx------  1 jsmith  nobody     375 Feb 16 16:47 fixit
```

Figure 5.3: Typical directory list appearance

line	::=	*path perm-specs*
path	::=	**string** | **string** `'/*'`
perm-specs	::=	*perm-spec* ';' *perm-specs*
	|	*perm-spec*
perm-spec	::=	*role-or-user permissions*
role-or-user	::=	**string** | `'*everyone*'`
	|	`'USER:'` **string**
permissions	::=	*permission* ':' *permissions*
	|	*permission*
permission	::=	FR | FC | FW | FD | FX | DC | DL | DR | LC
	|	F={R|C|W|D|X}+
	|	D={C|L|R}+

Figure 5.4: Syntax for role/permissions database

permission directly to an individual user. As in the example, this option is typically used to give users access to a home directory.

```
#
# The '#' character introduces a comment stretching to the
# end of line.
#
# The following roles exist: sysadm manager clerk guest
#
/                        sysadm     FR:FC:FW:FD:FX:DC:DL:DR:XT:LC
                                                          # Give full rights
                                                          # to sysadm

/                        *everyone* DL                    # Let everyone
                                                          # browse  FS

/sbin                    *everyone*                       # Deny rights
/usr/sbin                *everyone*                       # Deny rights

/usr/apps/dbms/audit.log  manager F=RCAD; clerk   F=CA
/usr/apps/dbms            manager F=RX;   clerk   F=RX
/home/jsmith              USER:jsmith F=CRWDX:D=CLR:XT:LC
```

Figure 5.5: Sample RBAC permission database

5.3 Implementation

The implementation was made using a publicly available NFS server as a starting point. The server software is in common use on Linux systems [SBS+95]. Our augmented server currently runs on a Pentium PC running Linux[3] 2.0.29 and is accessed over a standard 10 MBit/s Ethernet network.

As can be seen from figure 5.2, two main additions have been made to the original NFS server. One addition deals with the assignment of users to roles and the management of role activation and deactivation. The other addition is responsible for implementing the new, role-based, access control semantics as described in section 5.2.

5.3.1 Implementing roles

In order to use the concept of role, there must be a an administrative tool to *define* roles, it must be possible to *assign* roles to individuals, and there

[3]For an introduction to Linux, see for instance [Bok95, Yag96, Lin]

has to be a mechanism whereby roles may be *activated*. As stated initially, this task would normally be handled by an underlying security infrastructure. However, for the purpose of demonstration, we have implemented a client program that allows a user to communicate with a role state server adjacent to the NFS server. Using the client program, the user can activate and deactivate roles according to the assignments made in the **User/Role database**. This database is a simple text file that for each role lists which users may activate it.

5.3.2 Introducing RBAC semantics in the NFS server

By definition, any NFS server is built around the functions implementing the services defined in the protocol. We have introduced a two tier enforcement of RBAC in the server. The first tier delegates access control to the client machine by assigning suitable file attributes to referenced files. In many cases this is sufficient in order for the client machine itself to enforce access control. In the second tier, a check is made upon each service request to determine whether performing the request is allowed or not given the set of active roles for the current user. The modifications we have made can be summarized as follows.

Role establishment. In the original server, the first step taken when a service function is called is to authenticate the file handle, the calling machine, and the calling user. To this sequence we have inserted code that establishes a set of *active roles* for the duration of the call.

File attribute assignment. All read accesses to file attributes go through one function. Before returning from this function we have inserted a filter that clears out the access mode bits and substitutes a mode that is derived by combining (using bitwise or) the *necessary access modes* (see table 5.3) corresponding to each permission granted for the file to each active role. We also preserve the original value of the execute bit.

Access control at service request time. Here, a categorization into three categories can be made

1. No check. The `getattr`, `lookup`, and `readlink` services fall into this category, together with the unused or obsolete services.
2. Access permitted after check. The majority of NFS functions belong here. We have inserted code that checks if the requested access is permitted considering the set of active roles and the target path for the request. If this test fails, execution of the function is aborted and an access error status code is returned.

3. Access disallowed for all calls. Currently, for reasons set out in section 5.2.3, only the `link` call is disallowed.

Overall, the changes made to the original server are small. About 500, out of a total of about 13,000, lines of code were added or modified.

5.4 Results

We have tested our implementation on two Linux machines communicating with each other. One of the machines is a 200MHz Pentium processor workstation while the other is an older laptop machine with a 66MHz 486DX2 processor.

Some simple experiments were carried out to assess the impact our modifications have on performance. Two identical directory trees were created on each of the machines. The trees contain a total of 2,876 files and 115 directories. The built-in *time* command of *tcsh* was used to time two different commands performed on the directory tree. The first command was a full recursive listing of all files using the Unix command "`ls -lR`". To execute this command, the attributes for every file must be fetched from the server. The second command was designed to read about 6 Mbytes of data from the directory tree. This was achieved using the *GNU tar* command. In both cases the output was discarded by directing it to `/dev/null`. The operations were carried out using both the unmodified and the modified server, and in both possible directions. To make sure the servers did not use any information cashed from previous calls, the servers were restarted and remounted before running each command.

Table 5.4 shows the results from the "list" experiments and table 5.5 contain the results from the "read" experiments. Each entry contains the total time taken to execute the command. All values are computed means from three test runs.

	Server running on	
Server version	Pentium PC	486 PC
Original	22.33	46.0
Modified	24.98	1:49.0
Increase	12%	236%

Table 5.4: Execution times for `ls -lR` command

Server version	Server running on: Pentium PC	486 PC
Original	24.89	1:02.7
Modified	37.87	3:30.0
Increase	52%	339%

Table 5.5: Execution times for `tar` command

From the results we see that our modifications incur a rather heavy performance penalty that is relatively larger when the server is running on the machine with a slower processor. Even though the values shown for the faster machine seem to be acceptable, it is obvious that our rather straightforward implementation needs further refinement. For instance, information about active roles and permissions assigned to roles are currently kept in simple disk files. Here caching and use of in-memory databases can probably reduce the number of necessary disk accesses significantly. Also, our modifications to the original server are not primarily written with performance in mind. A tighter integration between base code and modifications is very likely to improve performance.

5.5 Related work

An interesting parallel to our work is the *Transparent Cryptographic File System* (TCFS) developed under the direction of Cattaneo and Persiano [CP96]. In the TCFS, the same basic NFS server that we use has been used to create a server that provides a file system offering transparent encryption of data on disk. The design goal for the TCFS is to provide a file system that guarantees that files cannot be read by anyone else than the legitimate owner, thereby excluding communication lines eavesdroppers and the super-user.

It is encouraging to note that Cattaneo and Persiano report that the performance of their server is close to that of commercial servers.

5.6 Conclusion

In this chapter we have shown how an NFS server can be modified to enforce role-based access control. We have demonstrated the feasibility of our main idea. However, performance needs to be improved. Nevertheless, we believe the enhanced server can be used to bring RBAC to a large number of existing systems and applications, without the need for modification of existing software.

Chapter 6

Some security techniques

This chapter provides a more thorough introduction to cryptography, authentication, and access control than was given in chapter 2. For in-depth studies of these areas we refer to some books in the area. *Security in Computing* by Pfleeger [Pfl97] presents a general overview of the area. Schneier's *Applied Cryptography* is a comprehensive volume on cryptography that also includes code examples [Sch95]. Salomaa gives an easy to read, but yet comprehensive, introduction to public-key cryptography in general in [Sal90, Sal96]. In her doctoral thesis, *Digital Signature Schemes*, Pfitzmann covers in depth the theory and use of digital signatures [Pfi96]. In *Authentication Systems for Secure Networks*, Oppliger surveys authentication frameworks in use today [Opp96].

6.1 Cryptography

Modern networks are vulnerable in that an intruder can often relatively easily gain access to a communications link and eavesdrop on communications.

An intruder listening in on communications could learn passwords or other information that enable him to pose as a legal system user. Other intruders might be satisfied with only stealing information flowing in the network, or might alter or replace it to gain an advantage, or cause damage to a competitor. It might, for instance, be tempting to add two or three zeros to an amount of money being transfered electronically between two banks, especially for someone having access to the receiving account.

There are however countermeasures to these threats. These measures have in common that they all build upon *cryptographic* techniques. There are two main cryptographic operations that can be used to provide secure network communications. A message that is transmitted can be *signed* and it can be *sealed.*

The signature of a *signed* message is constructed as a checksum that depends both on the contents of the message and on a cryptographic key. If the message has been changed in transit, the signature will no longer match the message, and the alteration can be detected.

When a message is signed, the contents of the message are not changed and can thus still be eavesdropped upon. To protect the contents of an electronic message from prying eyes, the message is put into the electronic equivalent of a sealed envelope. In the electronic context, *sealing* means to make the message itself illegible to anyone but the intended recipient by encrypting it.

An *encryption algorithm* is a mathematical function that takes as input a message, often called *plaintext* (or sometimes *cleartext*), and a *key.* The output of the function is a new message called *ciphertext.* Given the ciphertext, it should be infeasible to derive the clear text without the key, even if the enciphering function is known. Neither should it be possible to deduce the key even if both clear and ciphertext are known.

Cryptographic methods can be divided into two areas according to how the cryptographic keys are used. *Symmetric key* cryptography requires the *same* key to be known by both the originator and the recipient. In *asymmetric key* cryptography, keys come in *pairs*, where the originator has one key and the recipient the other. For secrecy to be maintained, it is paramount that the key needed for deciphering a message is kept secret.

6.1.1 Symmetric key cryptography

Ever since ancient times various kinds of codes and ciphers have been used to protect the confidentiality of sensitive messages should they fall into wrong hands. These methods have in common that the original message is transformed in some way using a reversible process that incorporates a secret that is known only to the communicating parties.

In encrypted electronic communication the shared secret is a key. The key is usually just a large number, sizes[1] of 40-128 binary bits are common for symmetric ciphers. When *symmetric* encryption algorithms are used, the same key is used both to encipher and decipher a message. The main advantage of using this technique is that very fast algorithms and even hardware implementations are available. This means that communications can be encrypted without imposing a substantial overhead.

There are also some disadvantages associated with symmetric key cryptography. The fact that the key must be available for both originator and

[1]The larger the size of the key, the larger becomes the space of possible keys, and the more difficult it becomes to guess a key or determine one by an exhaustive search.

the recipient means that it somehow has to be transferred, and that that transfer also has to be made securely. Another security concern is that it is necessary to trust all possessors of the key not to divulge it to a third party.

6.1.2 Asymmetric key cryptography

In contrast to its symmetric counterpart, asymmetric key cryptography requires the existence of a *pair* of keys. One, k_S, is usually kept secret whilst the other one, k_P, can be made public. The keys are so constructed that it is not feasible to derive the value of one key from the value of the other. Both keys have the same properties, so which one is kept secret and which is made public is of no consequence.

If we denote the encryption $\{M\}_k$, and the decryption $\{M\}^k$, of a message M, using a key k, the properties of the asymmetric keys k_P and k_S can be expressed as: $\{\{M\}_{k_P}\}^{k_S} = \{\{M\}_{k_S}\}^{k_P} = M$. An important property, however, is that for any key k of k_s and k_p it holds that $\{\{M\}_k\}^k \neq M$.

The great advantage of asymmetric key cryptography is that no secret key has to be transferred to another party. On the other hand, asymmetric key cryptography is computationally costly. For this reason a common use of asymmetric key cryptography is to securely exchange a symmetric one-time secret key between communicating parties. An additional advantage of this method is that asymmetric cryptography also can be used to authenticate the key being transferred (through digital signatures, see section 6.1.4).

6.1.3 Algorithms

Although many cryptographic algorithms have been proposed, only a few have received wide acceptance and use. An inherent problem in designing cryptographic algorithms is that such algorithms often rely on the conjectured intractability of performing certain mathematical transformations, for instance the factoring of large integers. Such conjectures are occasionally proven wrong, both due to new algorithms as well as an unforeseen increase in readily available computing power.

Here we give a brief account of some algorithms in common use today. Encryption algorithms belong to one of three main types.

Stream ciphers views the message to be enciphered as a bit stream. The encrypted message results from performing a bitwise exclusive-or operation between the message bit stream and a second, pseudo-random, bit stream. The generation of the second bit stream is controlled by the encryption key. The receiver deciphers the message by recreating the second bit stream and x-oring it with the encrypted message. The main advantage of stream cipher is that encryption and decryption can be implemented very efficiently.

Block ciphers function differently in that encryption is performed sequentially on fixed sized chunks of the message, a common block size is 64 bits.

Asymmetric algorithms have two keys, one which usually is kept secret while the other is publicized. A message that is encrypted with the secret key can only be decrypted using the public key, and vice versa.

Well known encryption algorithms are

RSA. This algorithm was presented by Rivest, Shamir, and Adleman ("RSA") in 1977 [RSA78]. RSA is an asymmetric algorithm that is widely used, primarily for authentication and key exchange purposes.

Diffie-Hellman. Named after its inventors, Diffie-Hellman is not an encryption algorithm, but a protocol that allows two parties to agree on a shared secret without any prior knowledge and without the need for transferring the secret itself [DH76].

DES. DES stands for *Data Encryption Standard.* This block cipher algorithm was endorsed by the United States Government as an official standard in 1977 and has since been revised several times. (The current specification is *Federal Information Processing Standards Publication 46-2* published by the National Institute of Standards and Technology [Nat93]). DES is one of the most widely used encryption algorithms in the world. However, even though DES implementations are available outside the United States, license to export software implementing DES from the United States is very rarely given.

RC2. This block cipher is a proprietary algorithm designed by Ronald Rivest for RSA Data Security. RC2 is designed to be a replacement for DES and also runs more efficiently than DES. Software utilizing RC2 that limits the key size to 40 bit is generally approved for export from the United States.

RC5. RC5 is also a proprietary algorithm developed by Ronald Rivest for RSA Data Security ("RC" stands for "Ron's Code" or "Rivest's Cipher") [Riv95]. In RC5, block size, key size, and the number of internal iterations (rounds) are parameterized. This parameterization makes the algorithm very flexible and allows it to be tuned to a wide range of different requirements on efficiency and level of security.

IDEA. The *International Data Encryption Algorithm* is a block cipher and is the result of a European development project. IDEA was first proposed by Lai and Massey [LM91]. The IDEA algorithm is patented in the USA and several European countries but can be used free of charge for non-commercial purposes.

RC4. This cipher is a proprietary and confidential stream cipher algorithm of RSA Data Security, designed by Ron Rivest.

SEAL. The *Software-optimized Encryption Algorithm* (SEAL) was introduced by Rogaway and Coppersmith in 1993 [RC93]. SEAL is a stream cipher designed for efficient implementation in software.

This list is by no means exhaustive. See for instance [Sch95] for a more complete enumeration and discussion.

6.1.4 Digital signatures

Alterations in a message can sometimes occur spontaneously due to noise or interference in the communications medium. To detect this, a message can be protected by attaching a checksum value with the property that the probability for a distorted message to have a correct checksum is very low.

However, such schemes cannot offer any protection against *deliberate* alteration of a message as anyone changing the message also can substitute the checksum. The checksum must be protected to form an *electronic signature* in order to assure that a deliberate alteration does not pass unnoticed.

Generation of an electronic signature goes through two stages. To reduce the size of the signature, a *message digest* is first created from the message to be signed. This digest is then encrypted to form the signature proper. Encryption ensures that only authorized parties can create a valid signature.

Message digests

A message digest is created using a *one-way hash* function. The digest usually has a fixed length such as 128 bits. Provided the digest has certain properties, signing the digest is as secure as signing the entire message.

There are several necessary properties a good message digest should have. Given a message and a digest, it should not be possible to find another message (that might be substituted for the original one) yielding the same checksum. Further, it should not be possible to find or construct two messages having the the same digest. If this were possible, it is conceivable that someone with fraudulent intent could construct two versions of, for instance, a contract, both having the same checksum.

A hash function having the above properties is said to be *collision free.*

Well known examples of message digest algorithms are **md4** and **md5** proposed by Rivest [Riv91, Riv92]. **SHA-1**, published by NIST (the National Institute of Standards and Technology in the USA), is a digest algorithm based on MD4 [Nat95, Sta94]. Dobertin has shown that MD4 is **not** collision free [Dob96]. Partial weaknesses have also been found in MD5 [dBB94].

These flaws cast some doubt as to the robustness of this entire family of algorithms.

Algorithms designed along new principles are being proposed (e.g. [AB96]). However, before new algorithms can gain acceptance, a long process of scrutiny and testing is necessary.

Signatures

Given a message digest, a digital signature is computed by encrypting the digest using the secret key of the originator. The recipient can verify the signature by computing the digest for the received message and comparing it to the digest obtained by deciphering the signature. Usually public key cryptography is used, although sometimes symmetric key encryption of the digest is also referred to as signing it. However, using symmetric cryptography provides no way of settling a dispute about message contents and thus does not really constitute a signature. A more proper term for a digest signed with using a symmetric key algorithm is *message integrity code*, MIC.

Encrypting the checksum with a key known only to the originator, or only to the originator and the recipient, in the case of a MIC being created, assures that the signature cannot be replaced.

A digital signatures scheme requires two functions, a signing function and a function to check the signature. Most schemes rely on the existence of two keys. One key, k_s, is used to generate the signature, $S = Sign(M, k_s)$. This key must be kept secret by the signer at all times, as anyone possessing it can produce a signature.

The second key, k_p, is made public. This key is used by anyone wishing to check a signature relative a message, thus the function $Check(M, S, k_p)$ returns true or false. The requirements on the functions *Sign*, *Check*, and the keys k_s and k_p are as follows

- For a message M and a signature S, generated with k_s, **only** k_p should yield a true result for $Check(M, S, k_p)$.
- There should exist no other key $k_s{}'$ such that $Check(M, Sign(M, k_s{}'), k_p) = True$
- It should be *computationally infeasible* to derive k_s from k_p.
- Without knowledge of k_s, it should be *computationally infeasible* to construct a signature *S'* for a message M, such that $Check(M, S', k_p)$ yields True.
- It should be *computationally infeasible* to find two messages yielding the same signature.

- For each pair of message digests M_{d1} and M_{d2}, if $M_{d1} \neq M_{d2}$ then the probability for $Sign(M_{d1}, k) = Sign(M_{d2}, k)$ should be smaller than the inverse of two raised to the number of bits in the signature. That is, no digest should be more probable than another.

To meet all of these requirements, a careful choice of the algorithms for message digest generation and encryption is necessary.

Assume we have signature S and a message M. Further, assume there is an originator that *alone* possesses a secret key k_s and there is a public key k_p *that is known to correspond to* k_s. We can then use *Check* to verify that:

1. the signature was produced using k_s
2. the signature was produced for the message M

As only the originator has access to k_s, we can conclude that the signature was made by her. Once the identity of the originator has been established we can turn to the contents of the message.

Applications

A very important application for electronic signatures is for an issuing authority to sign *certificates.* Certificates are basically documents that prove that an identity exists. In order for a certificate to be trusted it must be possible to verify its origin. This is done by letting the authority that creates the certificate attach a signature computed using a secret asymmetric key. Anyone can then check the authenticity of the certificate using the public key of the authority.

A complication here is that it can be difficult to obtain guarantees that a claimed public key indeed is the public key of the issuing authority. One approach to solving this problem is to create a hierarchy of certificate authorities where a public key for an authority is guaranteed by a signature by an authority on a higher level. At the top of this hierarchy there must be an authority that can be implicitly trusted (see further section 6.1.5).

6.1.5 Key distribution

As we have seen, cryptographic techniques are available to counter many security problems stemming from distribution. However, use of cryptography has its own problems. One example is that laws and export restriction can hinder international organizations in an effort to create homogeneous cryptographic environments. Another problematic factor can be the overhead and performance degradation caused by the introduction of encrypted or authenticated communications. There are other problems as well. We concentrate here on a very central issue, namely that of *key distribution.*

All cryptography relies on the use of keys. Furthermore, keys must be kept confidential in order for encryption to provide security. For symmetric cryptography, the problem of providing both communicating parties with a common, but secret, key, can be solved by using the Diffie-Hellman key exchange protocol or through the use of asymmetric cryptography. However, ensuring confidentiality and proper handling of asymmetric secret keys is a more difficult problem.

As asymmetric cryptography involves two keys, there are two sides of the key distribution problem. The confidentiality of the private (secret) key must be preserved and it must be possible to positively identify the owner of the public key for authentication purposes.

Conceivably, the integrity of a private key might be compromised at three stages:

Upon generation. If the system on which key generation takes places has been compromised, an attacker may eavesdrop and learn secret keys as they are created. To counter this threat, key generation should only be performed on a specially dedicated machine, unconnected to any network, to which physical access is limited to authorized persons only.

In very sensitive environments, key generation might be set up to require authorization from two or more security officers in concert. This reduces the risk for a fraudulent key being created in a "legal" way.

For the issue of keys that will be used to authenticate the identity of a person, it is paramount that the issuing authority takes adequate steps to prevent an imposter from obtaining a key using someone else's identity.

During distribution. Extraordinary care should be taken when transferring secret keys to nodes in a network. If such transfers take place over the network, an eavesdropper might intercept the message and then try to decrypt the message off-line. Due to the administrative overhead involved, asymmetric key pairs are changed relatively seldom. Thus it is not unlikely that a dedicated attacker, given a time span of months or years, might succeed in decrypting the secret key before it is changed. The most obvious countermeasure to this threat is to transport the secret key on a floppy disk delivered by courier. Once the key is installed on the target system, the disk should be carefully destroyed.

While in owner's possession. A basic assumption when using asymmetric private keys for authentication is that the rightful owner will not voluntarily divulge her key to others.

> Storing a secret key on disk in a workstation relies on the security provided by the operating system of the workstation. However, many operating systems are not secured against determined attacks. Furthermore, a private directory, where a user might place her secret key, is often part of a distributed file system. Hence, the secret key might end up being transferred over the network at each use. Even if secret keys are stored locally, this is not without problems as it forces a user to always use a specific machine, or to keep a secret key in several places.
>
> A solution to these problems is to store the key on a smart card. This alleviates the need for storing the secret key on disk and also allows the user to move between workstations. A smart card can also include a processor for performing encryption/decryption. This greatly improves security as there is no need for the secret key to ever leave the card, or indeed for there to be a way to extract the key at all.

The problems that arise when using and handling asymmetric public keys are of a different nature. The main issue here is to be able to positively associate a public key with its owner. This amounts to establishing an association between a key and an *identity*.

Except for very rare cases, a client cannot be expected to have first-hand knowledge of key/identity associations. Instead certificates that assert a key/identity association and that are *signed* by a *certifying authority*, CA, are used. If a client trusts the CA, it can trust the key/identity association found in the certificate. However, now another problem arises, how does a client know that the signature on the certificate is really that of the CA, and not that of an imposter? The obvious solution is to let the identity of the CA be certified by another CA, etc. The issue of what should be at the end of a such *certificate chain* is not yet fully resolved. Is it really possible to institute an authority that is *intrinsically* trusted by *everyone*? What would happen if the key of the top level CA was ever compromised?

Another important issue that must be resolved is that of *identity*. Assume Alice wants to send a confidential message to her associate Bob. If Alice knows Bob's public key this is not a problem. However, if Bob is a new acquaintance, Alice might not know the key. In this situation Alice might wish for a some sort of directory where she can learn Bob's key in way similar to how a telephone directory is used. X.500 is a recommended standard for a global, distributed, directory service originally published by CCITT [CCI88]. For information on X.500, see for instance [WR92, WRH92, URLb].

A problem with the *global name space* provided by X.500 is that Alice, in our example, must know Bob's *distinguished name*, which is a set of hierarchically structured information that includes fields such as country, organization, organizational unit, and name. If Alice does not have all this information, she cannot be sure to find the correct Bob she is looking for. Another problem with the X.500 name space is that it relies on organizations to make information about its members available. However, many organizations are reluctant to do this for integrity or business reasons, leaving the database incomplete.

Other infrastructures for making public keys available have been proposed. Rivest and Lampson have proposed *SDSI — A Simple Distributed Security Infrastructure* [RL96]. SDSI represents a move from a global name space to a private one. Each principal independently maintains a mapping between names and keys of other principals. A different approach is represented by ongoing work in the Internet community to define SPKI, the *Simple Public Key Infrastructure* [URLa]. SPKI does not have any direct mapping from individuals to keys at all. Instead, SPKI is an infrastructure designed for Cyberspace, where people meet and form associations, often without ever meeting in person.

6.2 Authentication

In many aspects of security, the identity of an actor who can perform, or has performed, some action, is very important. We usually do not want, for instance, anyone other than the account holder to be able to withdraw money from a bank account.

The process of establishing identity is called *authentication*. The most basic form of authentication is for an entity to trust the claims made by another entity. This will bring little in the form of security, but is nevertheless the method often used between people in daily life. However, both in computer applications and in daily life, trusting the word of another only applies where the consequences of doing so are not so grave.

In order to perform authentication, two things are necessary. First, an identity must be *unique*. A bank account belonging to Fred Smith should not be available to anyone who happens to be named Fred Smith, but just to *the* Fred Smith who opened the account. Hence, something more than just the string "Fred Smith" is required. In daily life people often use middle names or initials to help reduce any ambiguity as to who the individual is that their name denote. There will probably be fewer Fred Constantinus Xavier Smiths than Fred Smiths. However, uniqueness is not guaranteed. To accomplish uniqueness, authorities in many countries assign a unique number to each individual.

The second thing necessary for guaranteeing that only the intended Fred Smith can access the bank account is that there must be a way by which a person claiming to be *the* Fred Smith can *prove*, or make sufficiently plausible, that he really is. The process of establishing this proof is called *authentication*. In a bank, Fred Smith would probably exhibit an identity card of some sort.

Let us examine the properties of an identity card. Which requirements must it meet in order to be the proof of identity we desire? First of all it must be *difficult to forge*. If anyone could create an identity card with little effort we can as well take the word of Fred Smith when he makes his claim to his identity in the first place. Secondly, we must implicitly trust the entity that *can* produce the card not to issue fake ones. Again, if it is possible to acquire an identity card with any contents desired, it is of no value, even though it might be next to impossible to forge. Finally, it must be possible to associate positively and uniquely the card with a certain individual. A card that anyone can use once obtained, for instance by finding a misplaced one or by outright stealing one, is also of little value. For the purpose of making this association possible, most ID cards use a combination of a picture and a signature of the legitimate bearer. The looks and signature of a person have the advantage that they are always present. However, it is not entirely impossible to steal them. A cunning person might through forgery skill and clever disguise be able to use the ID card of another person. Still, for most daily life situations, ID cards are deemed sufficient. This is an example of how benefits from a security measure must always be weighed against other factors. It would, for instance, be possible to achieve a more certain association between ID card and bearer if the card also included a finger print. However, even though it is perfectly possible to check a finger print electronically, it would be awkward and would require extra equipment and software. A tradeoff has to made between the benefits of introducing the extra security measure and the expenses and other negative consequences incurred by doing so. A bank, for example, might not wish to *seem* not to trust its customers.

6.2.1 Computer authentication

A natural question is if it is possible to create the equivalent of an ID card for use in computerized communications? The answer is yes, but it turns out that things are a bit more difficult than the simple issue of some document.

Figure 6.1 shows the main entities in any authentication procedure. There is a *principal* that claims an *identity* and there is a *target* that must decide whether or not to believe in the claim. The means for making this decision are the *credentials*, the electronic equivalent of an ID card. The target believes the claimed identity to be authentic if the following conditions are met:

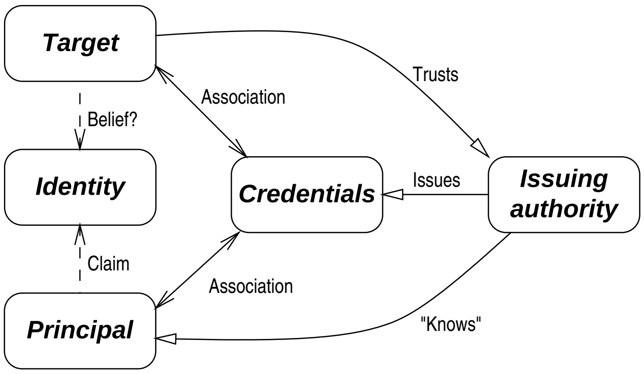

Figure 6.1: Authentication. A target authenticates a principal's identity by validating credentials and associating credentials with the principal.

1. The target *trusts* an issuing authority to issue *genuine credentials*. That is, credentials are only issued for *principals known* to the authority.
2. Issued credentials are *unforgeable*, that is, it must be possible to ascertain that none other than the issuing authority could have produced the credentials.
3. The target can *validate* the credentials as genuine.
4. The target can *positively associate* the credentials with the principal.

In the following we discuss each of these conditions in more detail.

6.2.2 Issuing authority

Embodied in the concept of an authority is a component of trust. It would be enormously impractical for every application to have first hand knowledge about all principals. Instead, we introduce an authority that we believe in and trust to have this knowledge. When it is necessary to know something about a principal, an application takes the word of the authority.

To earn the necessary trust, all possible safeguards must be put on how credentials are issued. For instance, policies for a *Certifying Authority* (CA) that issues electronic credentials, usually requires that physical access to the machine where credentials are created is restricted to a few specially authorized persons. Furthermore, policy can dictate that the pass phrases necessary to operate the machine must be known only to one half by any one person, and that written copies of such phrases can only be stored in a sealed container in a secure place. In addition, such policies must include precise conditions for when a set of credentials can be issued, for example only to individuals personally known to two of the authorized operators of the authority.

6.2.3 Credentials

The credentials themselves must meet two requirements: to be unforgeable and that they can be positively and uniquely associated with the intended principal.

Digital signatures can be used in order to meet both these requirements (digital signatures are discussed in some detail in section 6.1.4).

By letting the issuing authority electronically sign issued credentials, it is possible to verify that credentials originate from the proper place and not from someone else.

6.2.4 Association

Identity alone is not sufficient to identify the originator. Identities are public. For instance, if Alice sends her identity to Bob, there is nothing to prevent Bob from using the identity Alice, perhaps while communicating with some third entity. There has to be a way in which it is possible to uniquely tie a given identity to some specific entity. In simple terms, there has to be a way of proving you are who you claim to be. For ID cards, the means for doing this are twofold, the card contains the photograph and the signature of the person identified by the card.

In a computer system, a principal can be associated with a set of credentials using a digital signature scheme. In this process, a special signed message is created where an authority places the identity of a principal and a public key k_{p_p} that it *knows* corresponds to a secret key k_{s_p} in the possession of the principal. As explained in section 6.1.5, this type of message is commonly referred to as a *certificate*. Besides the identity, a certificate can include other credentials as well.

A principal can associate herself with a certificate by proving she possesses the secret key that corresponds to the public key present in the certificate, for instance by signing a message with it. Then an authentication server can verify the message using the public key found in the directory certificate. A basic assumption for successful and secure authentication is that the entity identified in the certificate is the *only* entity in possession of the secret key.

However, far from all computer systems rely on digital certificates and signatures for establishing the association between principal and credentials. The most common method used today is passwords. Usually, the first thing a user logging into a computer system has to do is to provide her user identity. For instance, in a Unix system, the identity points out a set of credentials for the user that is represented by a line in the `/etc/passwd` file. A user provides proof of association with these credentials by providing a password that corresponds to the hashed version of the password present in the `passwd`-entry.

However, passwords often do not provide the desired level of security. Users tend to select passwords that are easy to remember but are easy to break. Password cracking programs for Unix systems are publicly available and usually succeed in guessing at least some passwords in an average user population just by using words and combinations of words from a dictionary. Furthermore, in many existing networks, passwords are sent in plaintext from client to server. For this reason, it is sometimes desirable to complement the use of passwords with other means for authentication or to use methods and authentication protocols that use passwords only to derive an encryption key that is used to protect authentication information sent over the network.

While a password is something a user *knows*, complementary means can also be based on something the user *possesses*. Common examples are smart-cards and cryptographic calculators. These implements suffer from the drawback that they may be lost or stolen. Other devices instead rely on *biometric information* such as facial features, fingerprints, retina patterns, and voice prints, that cannot be separated from the bearer. However, techniques like these all require specialized, and often expensive, equipment that can be difficult to interface to a particular type of computer used in an organization.

6.2.5 Validation

As set out previously, using the public key, an application can validate a set of credentials by checking the signature put there by the issuing authority. However, the question immediately arises as to how an application can *know* that the public key is really that of the authority, and not the one of a phony authority that has issued false credentials. The immediate solution is for the authority to also prove *its* identity, much in the same way as the principal is trying to do.

Using this scheme, one authority is vouched for by another, that is in turn vouched for by yet another, etc. This of course ultimately leads to an infinite regress. For this reason it is necessary that there exists a *top level authority* that is *intrinsically trusted* and where the public key is distributed in such a way that there can be no doubts about its validity.

6.2.6 Conclusion

In this section we have shown the principles for how an application can establish the identity of a principal using the electronic equivalent of an ID card. As we have seen, the basic problem is to be able to tie, uniquely, an identity to an entity. In the ID card example, this is achieved by tying the identity stated on the card to something that is *unique* and *inseparable* from the individual (appearance and signature). In the other familiar example mentioned, that of logging into a computer system, the most common way of corroborating that the claimed identity is the real one is for the user to provide a password. This works on the principle that *only* the legitimate user knows the password and that the password cannot be easily guessed. The system must somehow have been told how to associate a password with an identity. From these examples we can identify two principles for corroborating the identity of an entity. The first is to rely on something the entity *has*, (appearance, ability to write a signature). The other principle is to rely on something the entity *knows*, such as a password.

It is important to note that the complete and successful authentication is dependent on many links in a chain. As for any chain, the authentication procedure can only be as strong as its weakest link.

6.3 Access control

While proper authentication guarantees that acting entities in a system really are who they claim to be, this in itself does not say anything about what these entities may, or may not, do. To enforce security policies to that effect we need *access control*.

Access control can be found in many places and implemented on many levels. Most operating systems have access control mechanisms on the file level. Many database systems allow access control rules to be specified for how the data in the database may be accessed. There are also many examples of applications that implement their own access controls, independently of other mechanisms that may exist in a system.

The most basic access control policy is to allow full access to anyone properly authenticated, and to deny access completely otherwise. This model is perfectly suitable in some environments, but is too unsophisticated for many others. There are several approaches to how access control can be organized in order to best express and implement underlying security policies. Later on we shall look at some of the most important of these.

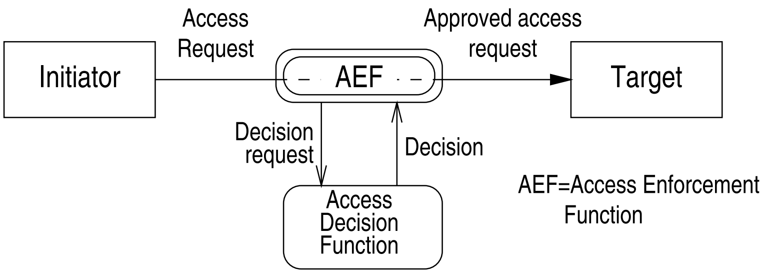

Figure 6.2: Abstract access control model

6.3.1 General principles

The International Organization for Standardization (ISO) together with the International Electrotechnical Commission (IEC) has published a suite of documents specifying and standardizing various security-related frameworks. Figure 6.2 is adapted from [ISO96] in this suite. The figure shows an abstract view of access control using the standardized terminology. The *initiator* is the person or program that attempts to access some *target*. When access control is in place, an *access enforcement function* (AEF) is interspersed in the path of the *access request*. The AEF calls upon an *access decision function* (ADF) to get information about whether an access request should be granted or not.

The ADF may use information from a number of sources in order to compute its result. *Access control information* (ACI) may be associated with

The initiator. Initiator ACI may be attributes such as the (access control) identity of an individual, information about group membership, information about active roles, and sensitivity and integrity markings.

The target. Examples of target ACI are target access control identity, sensitivity and integrity markings, and identity of a container object holding the target.

The access request. To access a target is to perform some kind of operation on it. Access control information for operations can include things such as classification (e.g. read, write) and an integrity level required for use of an operation.

A request operand. Operands other than the target may be involved in an operation, for instance information to be written to a file. Such operands may also have ACI-like integrity or sensitivity markings.

In addition, the ADF may also take **contextual information** into account when making its decision. Examples of contextual information include the time of day, the location of the initiator, the route a request has traveled, the strength of authentication used, and system status. Contextual information of this kind can be used to enforce policies restricting access to certain time periods, secure terminals, secure communications channels, or to strongly authenticated initiators.

6.3.2 Getting a working system

In order to create an access control system in a real system, the abstract entities described above need somehow to be instantiated. An initial step is to establish representations and data structures for storing access control information in the system. Such representations are highly system dependent, and must be carefully crafted to allow access control policies to be expressed in the desired way. There must also be implementations of an AEF and an ADF.

Given these basic building blocks, the operation of an access control system can be divided into two main parts. One is the operation of the system itself, the granting or denying of access requests as required. However, a prerequisite for the proper workings of this is that ACI has been assigned correctly to the various entities in the system. This management and administration of the access control system itself form the second main part.

Management of an access control system is performed by a *Security Domain Authority* (SDA), or by agents of the SDA that have been delegated the authority to perform management operations. Typical management operations include things such as installing, changing, and revoking ACI to initiators, operations, and targets. This also includes the important step of *defining* ACI attributes and their *value domains.*

Once ACI is in place, the system can begin operations. The main difficulty to solve when realizing an AEF and an ADF is how to make ACI available to these entities in a secure manner. In order for the ADF to make the correct decision, it must be unequivocally certain that the ACI it has to evaluate really has been properly assigned to the entities involved in the operation. For this purpose, the concept of *bound* ACI is introduced. The process of binding provides the necessary proof of association between some ACI and an entity. For any access request, there will be ACI bound to the initiator, the target, and the request itself.

6.3.3 Classification of access control systems

The framework for access control described in section 6.3.1 is a very general one within which most systems can be described. By looking at the way ACI is used, it is possible to classify access control systems using just a few main schemes. It is also possible to classify access control systems according to underlying access control policies.

ACI usage schemes

We briefly describe here four ACI usage schemes. It should be noted that most access control systems allow for more than one scheme to be used. The usage schemes we describe are:

Access Control Lists. Access control lists (ACLs) is a widely used scheme. An access control list is associated with a target (and becomes target-bound ACI to the ADF.) The list consists of initiator/operation pairs naming an individual, group or role that may perform a specified operation on the target. When using ACLs, necessary initiator-bound ACI is an individual, group our role identity.

ACLs provide for a very finely grained level of access control. However, if the number of possible initiators is large, or if the set of initiators varies frequently over time, maintenance of the lists can become very demanding.

Capabilities. Capabilities are self-contained initiator-bound ACI that name a target and a set of permitted operations on that target. A capability is analogous to a parking permit. In the same way a parking permit shows that the holder may park vehicles (an operation) at some specified parking space (a target), a capability shows the ADF the operations the initiator may perform on the target.

Capabilities are issued and bound to an initiator by an SDA (security domain authority). Target-bound ACI contains information about existing SDAs and what operations each SDA may authorize.

Labels. In a label-based access control scheme, security labels may be assigned to initiators, targets, operations, and data in transfer. The ADF compares the labels of the entities in a request and makes its decision according to some set of rules.

As an example, consider a system where all information is assigned one of the labels **Unclassified**, **Confidential**, and **Secret**. Users are also assigned corresponding labels, usually called clearances. ADF rules can then state that someone with **Secret** clearance may read information labeled **Unclassified**, **Confidential**, and **Secret**. However, everything written by a user with **Secret** clearance must also be labeled **Secret**. Likewise, a user with **Confidential** clearance may read **Confidential** and **Unclassified** material, but can only write **Confidential** or **Secret** data.

Labels can be arbitrary complex entities and are often structured so as to facilitate comparison.

Context based access control. To implement some policies, it is desirable to take into account contextual information pertaining to an access request. Such information can be the time of day, the location of the initiator, the route which the request has traveled, or the current system load or status. Often such restrictions are imposed generally for all initiators.

In order to enforce contextual rules, the ADF uses a context control list. This list contains pairs of context specifier and allowed operations. The ADF can obtain context information both from initiator-bound and target-bound ACI, as well as independently.

It is important to note that these schemes are only examples of how the inner workings of an access control system can be structured. The use of one or the other scheme has no direct relation to what access control policies can be expressed and implemented. However, the ease of policy implementation varies for different policy/scheme combinations.

Access control policies

In this section we look at some common overall access control policies. We also relate these policies to those access control schemes which are most often used to implement them.

Mandatory access control. The name Mandatory Access Control, MAC, is perhaps something of a misnomer as access control should never be "optional". However, MAC is a long used and widely accepted term to denote an overall policy where all ACI is maintained by a central SDA. This centralized approach has advantages in that it gives a common view over how access control policies are implemented.

The label-based access control scheme is often used to implement MAC.

Discretionary access control. While it can be said that for MAC, the SDA "owns" all resources, *Discretionary Access Control*, DAC, introduces a concept of *resource owner*. Under DAC, it is the resource owner that manages ACI for a resource. A well-known example of DAC is the

Unix operating system where each user can decide the availability of her files and directories to other users. This approach greatly reduces the need for centralized access control administration and provides great flexibility. However, under DAC it is often difficult to enforce overall access control policies as one must rely on each and every resource owner to individually implement policies as they apply to the resources owned by her.

Discretionary access control is often implemented using the access control scheme where each target-ACL is managed by the target owner.

Role-based access control. Role-based access control (RBAC) is a way to express access control policies using organizational roles as an origin. The basic idea is that when an individual acts within an organization she does so in the capacity of a *role* defined within that organization. Using roles, and relationships between roles, an organization can be described abstractly. Access control information is then added to the abstract organizational model, i.e. authorizations are assigned to roles. In this way access control policies can be completely expressed in a way that does not involve specific individuals. In a second step, individuals are assigned to the roles in which they are allowed to act. In a third step, a user who wishes to perform some task activates a role (to which she has been assigned) appropriate for the task and thereby indirectly obtains the permissions given the role.

Role-based access control can be implemented using several schemes. The most straightforward scheme is to use access control lists naming roles as initiators. In distributed systems a capability-based scheme is often used where users obtain a set of capabilities when a role is activated. RBAC models often also include constraints on how roles may be used, calling for the contextual access control scheme to be incorporated in the implementation.

6.4 Conclusion

In this chapter we have given an introduction to selected computer security techniques and concepts. Security for distributed systems is still an open area for research and development. Perhaps the lower-level methods, different cryptographic methods and so forth, can be considered relatively well understood. However, on the intermediate and high system levels we see still the need for much further work, for instance in creating functioning key distribution infrastructures and other trusted third party services.

Chapter 7

Survey of secure distributed systems

In this chapter we look at some existing architectures, protocols and frameworks designed with security in mind. As we shall see, security measures can be implemented at different levels. At the lowest level we discuss *IP level security* in section 7.1 and *Secure RPC* in section 7.2. The *Secure Socket layer*, *Secure Shell* and *Secure HTTP* (7.4, 7.3, 7.5) are protocols on somewhat higher levels. Finally, *Kerberos*, *CORBA* and *SESAME* (7.6, 7.7, 7.8) are higher-level security frameworks.

7.1 IP level security

A suite of three Internet Request for Comments (RFC) documents specify an architecture for how the Internet Protocol (IP) for network communications can be augmented with services for authentication, integrity and confidentiality. Of these documents, [Atk95c] provides an architecture overview while the other two specify in detail the proposed new services [Atk95a, Atk95b]. The specified services can optionally be included in IP version 4 (IPv4, the protocol in common use on the Internet today) implementations. For the forthcoming IP version 6, implementation of these services is mandatory.

As IP is a low-level protocol unaware of higher-level protocols or applications that use it, security services are provided between hosts only. It is not possible to achieve user- or application-specific protection using IP-level security.

To implement IP-level security, two extensions to the IP protocol are defined. The first extension specifies an *IP Authentication Header* (AH)

[Atk95a]. An encrypted checksum computed is placed in this header. The checksum is computed using a message digest algorithm over the rest of the fields in the IP datagram. Upon receipt, the remote host checks the validity of the IP datagram using the information in the Authentication Header. If this authentication fails, the datagram should be discarded. Information about the failing packet is also entered into the system log.

The second extension specifies a header named *IP Encapsulating Security Payload* (ESP) [Atk95b]. An ESP IP datagram contains a set of unencrypted IP headers, the ESP header, and a block of encrypted data. There exist two operating modes for ESP. In *tunneling mode* the encrypted data contains a complete IP datagram. Upon receipt, the target host decrypts this datagram and processes it as if it had arrived directly (conceivably the tunneled datagram could itself be ESP-enhanced). In the other operating mode, the encrypted data contains an upper-level protocol (UDP, TCP, etc) data frame.

To benefit from security-augmented communication, the target host must be made aware of which authentication and encryption algorithms and modes are used by the sender. A collection of such information is termed *Security Parameter Index* (SPI). An SPI is identified by a 32-bit number present in the AH or ESP header. SPI-identifiers in the range 0-255 are reserved and only assigned by the Internet Assigned Numbers Authority (IANA).

7.2 Secure RPC

The *Remote Procedure Call* protocol (RPC), specified by Sun Microsystems, is a protocol running on top of a transport layer protocol such as TCP or UDP. To the application programmer, RPC hides the details of network communications by making it possible to transparently call functions that are executed on a remote machine [Sun88]. A well-known application that uses RPC is the *Network File System* (NFS), also specified by Sun [CPS95].

In the basic configuration, RPC requests in a Unix environment contain credentials giving the user and group IDs of the caller. However, these values are not authenticated, making RPC vulnerable to impersonation attacks. In the case of NFS, this could mean that an intruder could get unauthorized access to entire file systems.

To amend these shortcomings, *Secure RPC* implements strong authentication. In secure RPC communications, the client and server must first agree on a symmetric session key. This can be achieved using asymmetric key cryptography where the client has a secret key and where the corresponding public key is known by the server (and known to belong to the client). Once the session key has been established, RPC packets passed between server and client contain a field with the current time, encrypted using the session key. As the session key is known only by the client and the server, this achieves mutual authentication.

7.3 SSH

SSH is an acronym for *Secure Shell* and is the common name of a suite of programs designed by Tatu Ylönen [Tat97]. The main program, **ssh**, is designed to be a direct replacement for the **rsh** (*Remote Shell*) program found on many Unix systems. SSH provides strong authentication and eliminates many known vulnerabilities. For instance, passwords are never sent over the network in clear text using SSH. In fact, using SSH, all network communications are automatically and transparently encrypted. Replacements for other utilities in the "**r***"-family of commands, such as **rlogin**, **rsh**, **rcp**, and **rdist** are also provided with **ssh**.

Work is underway within a working group of the IETF (*Internet Engineering Task Force*) to standardize the SSH protocol.

7.4 SSL

The *Secure Sockets Layer*, SSL, is a protocol proposed by Netscape Communications to provide authentication, confidentiality and integrity services for Internet communications. Version 3.0 of the protocol still has the status of *Internet Draft* and has thus not yet been finalized. However, Netscape Communications have already incorporated support for the protocol into their products.

SSL is a client/server protocol that is designed to be interspersed immediately above the transport layer in the IP protocol suite, as illustrated by figure 7.1. An advantage of SSL is that higher-level protocols, and applications based on them, need not be aware of the existence of SSL.

The SSL protocol is in itself layered, the *Record Layer* interfaces SSL to an underlying stable transport protocol, such as TCP. Higher-level protocols within SSL exist to establish and use SSL *sessions.*

7.4.1 Record layer

Communications utilizing SSL take place in *connections.* Associated with each connection is a number of attributes that describe the current *connection state.* State information is kept at both end points. Connections between two entities are always established within the context of an SSL *session.* The state attributes associated with a session apply to all connections within that session. Several sessions may be active at the same time.

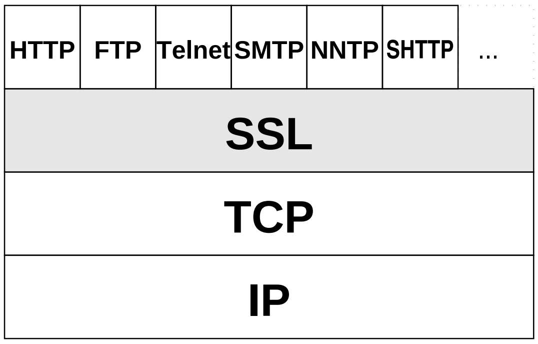

Figure 7.1: Relative placement of SSL in the IP protocol suite.

A record that is to be transmitted through SSL is first compressed and then protected. Protection is afforded by the addition of a *sequence number* and a *message authentication code* (MAC) to each record before it is *encrypted.*

To compute a MAC, MD5 or SHA can be used. For encryption either a *stream cipher* or a *block cipher* can be applied. The supported stream cipher is RC4 and supported block ciphers are RC2, DES, and FORTEZZA (see section 6.1.3 for information on algorithms).

The session state contains information about which method should be used in each of these transformation stages. Any or all of the transformations can be the identity transformation. In the latter case no protection is offered.

Connection state attributes contain parameters, such as cryptographic keys, that are used when applying the transformations specified in the session state.

7.4.2 SSL internal protocols

The SSL specification defines a number of protocols internal to SSL that exist on top of the record layer. These protocols are

Handshake protocol. The responsibility of the handshake protocol is to establish a session between a server an a client and in doing so perform necessary authentication and exchange of encryption keys to be used for further communications. A special *hello message* is used by the client to confer to the server a set of preferred combinations of key exchange algorithm, MAC method and bulk encryption algorithm. A set of possible such combinations, called *Cipher Suites*, are defined.

The hello message also includes a list of supported compression methods. In response to a client hello, a *server hello* is sent. This message informs the client of a *session id* for the session and which cipher suite and compression method the server has selected to use from the alternatives provided by the client.

In addition to the hello messages, the handshake protocol also specifies how certificates and session keys are to be exchanged.

A client may send a client hello message upon request from a server, or on its own initiative to establish or re-negotiate the parameters for a session.

Change cipher spec protocol. The exchange of hello messages and other information causes a *pending session state* to be established. The current state of an entity is replaced by the pending state upon receipt of a *change cipher spec message*.

Alert protocol. The SSL *Alert Protocol* specifies how error messages can be sent between communicating parties. The most common action upon sending and receiving an alert message is to close the connection.

Application data protocol. The application data protocol is not specified further than to provide an interface to the record layer through which application data can be sent and received, and transparently subjected to compression and encryption according to the current session state.

7.5 S-HTTP

The *HyperText Transfer Protocol* (HTTP) is one of the foundations upon which the World Wide Web is built. HTTP itself has only rudimentary provisions for securing web communications. To amend this situation a working group of the Internet Engineering Task Force (IETF) is currently working to design *Secure HTTP* (S-HTTP), a protocol that aims to bring security mechanisms to HTTP servers and clients.

S-HTTP is designed to be flexible and to be able to make use of a wide variety of cryptographic algorithms and working modes. Exactly what protection (integrity, signing, confidentiality) is used for a connection is determined by an automated negotiation processes that takes its input from user preferences and what capabilities are present in the client and server.

The basic working mode of S-HTTP is to encapsulate an ordinary HTTP message within an S-HTTP message. An S-HTTP message consists of a number of S-HTTP specific headers that state which security enhancements have been applied to the message. This information is necessary for the recipient to successfully retrieve the original HTTP message. The HTTP message itself is signed and/or encrypted as required. A client that receives an S-HTTP message first reverses any encryption and performs a check of message signatures. If successful, the encapsulated HTTP message is extracted and passed to the browser.

S-HTTP is not restricted to transfer HTTP messages; any message type might be encapsulated.

7.6 Kerberos

Kerberos is a protocol for strong authentication in a network environment originally developed as a part of the Project Athena at the Massachusetts Institute of Technology. Neuman and Ts'o give a more complete overview of Kerberos in [NT94]. The detailed and rather lengthy official specification is published as an Internet RFC [KN93].

The core entity in a Kerberos system is the *Authentication Server*, AS. A principal who wishes to use Kerberos services needs to share a *secret key* with the AS. The set of principals that are serviced by one authentication server is called a Kerberos *realm*.

When a client wants to authenticate its identity to some service provider it acquires a set of credentials from the authentication server. In Kerberos terminology, the set of credentials is called a *ticket*.

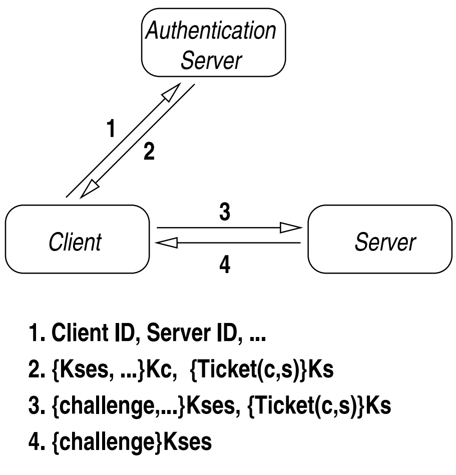

Figure 7.2: Basic Kerberos authentication

Figure 7.2 contains a schematic illustration of the Kerberos message exchange. When requesting a ticket, the client presents its own name to the authentication server together with the name of the service to which it wants a ticket. In response, the AS generates a reply that contains two parts, the first part contains information to be used by the client and the second part is the ticket that is to be passed on to the service. The client part is encrypted using the key shared between the AS and the client while the ticket is encrypted using the key shared between the AS and the service. Both the client information and the ticket contain a randomly generated *session key* for use in communications between the client and the service.

When the client initiates communications with the server, it encrypts the message using the session key obtained from the authentication server. The client also forwards the server ticket. Upon receipt of the ticket and encrypted message, the server decrypts the ticket to learn the identity of the client and to extract the session key. The session key is then used to decrypt the client message. If the server can successfully decipher the client message, it has also established the identity of the client, as the only other entity that has access to the session key is the client to whom the ticket was issued.

Sometimes, the client, too, wants to ascertain the identity of the server. To accomplish this, the client includes a random *challenge* in the original message. The server proves it is in possession of the session key by returning to the client the challenge encrypted under the session key. As the original client message contains more information than the challenge, the server must possess the session key in order to decrypt the message before the challenge can be extracted.

7.6.1 Ticket-granting server

A Kerberos client is often a system user. The secret key shared between a user and the authentication server is most commonly derived from a pass phrase known by the user. When the secret key is needed, it is generated from the pass phrase. However, it is undesirable to keep the secret key stored in the client for any longer than necessary. It is also inconvenient for the user to have to supply a pass phrase each time a new ticket is needed. To solve these problems, Kerberos uses a special server called the *Ticket-granting Server* (TGS).

As the name implies, the TGS grants tickets to clients. In order to use the TGS, a client must first obtain a *ticket-granting ticket* (TGT) for the TGS from the authentication server. However, once the TGT has been obtained, and a session key has been established between the client and the TGS, the client secret key is no longer needed, and can be forgotten. For further ticket requests, the client can use the TGS. The identity of the client is proven to the TGS through possession of the session key.

7.7 CORBA

The acronym CORBA stands for *Common Object Request Broker Architecture*. CORBA is foremost a platform for distributed, object-based, computing. Behind CORBA is the Object Management Group (OMG), a consortium of originally eight large international information technology companies. Today, OMG has over 500 member companies and organizations.

Within the CORBA architecture, the role of the *Object Request Broker* (ORB) is to provide connectivity between objects. CORBA Security Services [CHN+94, The95] are implemented inside the ORB, thus (to a degree) making them transparently available to communicating objects.

The specification contains a reference model covering *message protection*, *access control*, *auditing*, *delegation*, and *non-repudiation*.

Message protection is concerned with integrity and confidentiality of object invocation communications.

Access control in the ORB can be carried out both at the client and at the target upon object invocation. Access control is carried out using an access decision function according to the model specified in [ISO96] (see section 6.3).

Auditing is provided for in the specification both at the initiating and target side of the ORB. What information should be written to the audit trail is decided by a set of system audit policies together with specific application audit policies.

Delegation of privileges is often necessary when one object needs to call upon another object to perform its own services (the server becomes a client). The CORBA security reference model allows for a wide variety of delegation models.

Non-repudiation involves the creation of irrefutable evidence of some event taking place. In contrast to the previously listed security services, non-repudiation is under the control of the applications utilizing the services. Non-repudiation can be used for many purposes, common examples are non-repudiation of creation and non-repudiation of receipt. In the first case, a creator of a message that later denies having created it can be proved to be lying. In the second example non-repudiation services provide protection against someone falsely denying having received a message.

The reference model also defines the concepts of *security policy domain* (a grouping of objects to which the same security policies apply) and *security authority* (the administrator of a security domain).

The CORBA security specification goes on to specify in detail the services specified in the reference model.

7.8 SESAME

SESAME is a comprehensive framework for secure distributed environments. The SESAME project was initiated by the Commission of the European Communities, CEC, to demonstrate the usability of then ongoing standardization work with ECMA. After achieving the initial goals, the SESAME project was continued by three major European computer and software manufacturers; Bull, ICL, and Siemens Nixdorf. This work resulted in version 4 of SESAME in 1995 [PP95].

SESAME provides an infrastructure that allows secure client/server interactions over a network. From the point of view of an application, interaction with the SESAME system is through the Generic Security Service API [Lin97] (for more information on GSS-API, see section 3.4.1).

Figure 7.3 shows the main components of the SESAME framework. In a typical sequence of events, a user starts by identifying herself to the SESAME system through a special *user sponsor application*. The user sponsor forwards the user's request for authentication, along with the user's password, to the *Authentication and Privilege Attribute client*. This special program contacts the *Authentication Server* running on the *Domain Security Server* machine. If the authentication is successful, the AS returns a Kerberos style *ticket* for the *Privilege Attribute Server* (PAS).

The ticket for the PAS is later used by the *Secure Association Context Manger*, SACM, to obtain a *Privilege Attribute Certificate*, PAC, and a ticket for the *Key Distribution Server*. When a client application wants to initiate secure communications with a remote server, the application calls upon an initialization function within the GSS-API. The API in turn asks the SACM to provide keying information pertaining to the server to be called. The GSS-API initialization call returns a GSS *token* that contains a ticket for the server and an encrypted instance of the PAC. The token is passed to the server application.

Upon receipt of the GSS token, the server passes it to the target side SACM which extracts its components. The target SACM examines the service ticket and validates the supplied PAC using the *PAC Validation Facility*. If the PAC and ticket are valid, a local access control check is performed before a return token is generated. This token contains newly generated dialogue keys that are used for further secure communications. The token is passed back to the server application, that in turn returns it to the client application. The client application must then pass the returned token on to the GSS-API. A GSS-API *security context* can then be established between client and server. Among other things, the context contains information about dialogue keys and cryptographic algorithms to be used.

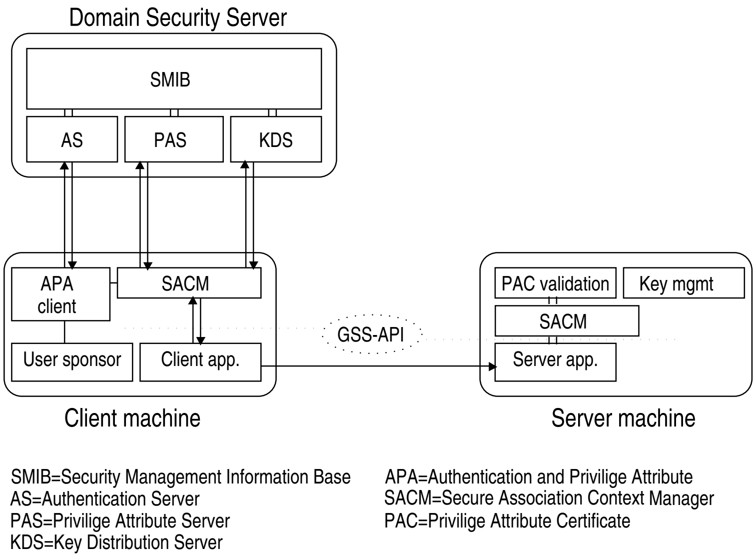

Figure 7.3: Main components of SESAME

The basic purpose of SESAME is twofold, to provide an authentication framework and to provide an access control infrastructure. In addition to this, the GSS-API also offers confidentiality and integrity services.

In SESAME, access control is primarily capability-based (see section 6.3.3). A PAC contains the effective privilege attributes of an identity. The attributes are computed using the authenticated identity and active role. A PAC is cryptographically signed by the PAS and includes the identity for which it has been issued. If the target application trusts the PAS, and is certain about the identity of the client, it can validate the PAC and then directly use the access control information there.

Chapter 8

Concluding remarks

Computer security is a highly faceted topic. In the first chapters of this thesis we have provided an introduction and overview of the area that we believe provides a good background for later chapters in the thesis, as well as for someone wanting to pursue their own studies in the area.

In the thesis the emphasis has been on security in distributed systems. Within this area we have presented a number of contributions of our own. Our contributions cover different sub-topics of the main area and by doing so also help to emphasize the diversity and size of the field of distributed systems security.

In the design for an object-oriented framework for secure distributed systems, we utilize the transparency provided by the underlying object distribution mechanisms to provide centralized monitoring and control over an otherwise distributed system. Such monitoring and control is often desirable but is not provided for in other distributed security frameworks. By using an object-oriented approach, the interaction between applications and the security framework becomes well-defined. This is very important as it allows existing applications to be encapsulated in objects that can interact with the new security infrastructure.

In the role representation framework we have presented, we lay out some foundations to how we can think about and represent organizational roles. These also serve as basic building blocks upon which to place the important concept of role-based access control. RBAC has many advantages, administrative and other, and we believe it has an important place in any security framework.

Development always brings with it conflicts between the new and the old. What is usually referred to as legacy systems can, by their mere existence and the investment they represent, often slow down or hinder the introduction of new technologies. For the case of distributed security a problem is that it can be difficult to fit existing applications into new security frameworks. Our object-oriented framework offers one solution to this problem. We also present another solution, on a different level, where we make an NFS server RBAC-aware. In this approach we place existing applications under role-based access control by modifying the underlying file system in which they operate. For the sake of demonstration, we have presented this as a "stand-alone" solution. However, in a real application our solution should be integrated into some more comprehensive framework.

8.1 Future work

This work serves as an introduction to a large and important area of ongoing research. In this section we point at some interesting avenues for further work and research.

Frameworks

We believe that the object-oriented framework architecture we have presented has several interesting properties. However, the framework needs further elaboration. The design of the centralized context manager object in particular needs careful consideration as any vulnerability here would have far reaching consequences. Issues such as safeguards and interfaces must be addressed, as must also performance and real-time properties.

Role-based access control

RBAC is intuitively appealing and certainly has an important future. There already exists the means for introducing RBAC into distributed systems. In fact, as RBAC is mainly a way of expressing access control policies, enforcing mechanisms remain about the same regardless of whether RBAC is used in a system or not.

There are, however, many other aspects of RBAC that we believe are important and that are not yet fully understood. We see that more work needs to be done on the representation of roles and role-based access control policies. This, for instance, involves issues such as hierarchies and inheritance. We also envision the need for a formal description framework inside which it is possible to model authorizations as well as various constraints. Ideally, it should be possible to use such a representation to verify a model instance, for instance to assure that it is free from conflicting rules. Another interesting application would be to be able to study the ramifications of proposed changes before they are carried out.

Another area of research that is important for the future spread and proliferation of RBAC is how RBAC can be efficiently deployed. For instance, we see the need for a number of efficient RBAC administrative tools for creating and managing the role structure itself, as well as for managing assignment of users and privileges to roles.

Here research is needed to see if it is possible to some extent to automatize the process of translating high-level access control policies into tangible rules expressed in terms of roles and resources.

Bibliography

[**Note**] *Some citations include Universal Resource Locators (URLs). These URLs have been verified to be correct at the time of printing. Due to the dynamic nature of the Internet, there are unfortunately no guarantees for how long an URL will remain operable. However, as long as a resource remains available electronically, effort and delays in obtaining a reference can be greatly reduced. For this reason we have, despite their limited persistence, decided to include a URL with citations where an electronic copy or information is available.*

[AB96] Ross Anderson and Bli Biham. Tiger: A Fast New Hash Function. In Dieter Gollmann, editor, *Fast software encryption: Third International Workshop*, pages 89–97. Springer-Verlag, 1996.

[Atk95a] R. Atkinson. IP Authentication Header. RFC 1826, August 1995.

[Atk95b] R. Atkinson. IP Encapsulating Security Payload (ESP). RFC 1827, August 1995.

[Atk95c] R. Atkinson. Security Architecture for the Internet Protocol. RFC 1825, August 1995.

[Att] Attachmate. PathWay Server NFS for OpenVMS. Product information, World Wide Web document. http://www.attachmate.com /OSG/NFSServerVMS/default.asp.

[Bid79] Bruce J. Biddle. *Role Theory – Expectations, Identities and Behaviours*. Academic Press, 1979. ISBN 0-12-095950-X.

[Bok95] S. N. Bokhari. The Linux operating system. *Computer*, 28(8):77–9, Aug 1995.

[CCI88] CCITT. *Blue Book, Volume VIII - Fascicle VIII.8, Data Communication Networks Directory, Recommendations X.500-X.521*. CCITT, 1988. ISBN 92-61-03731-3.

[CHN+94] S. Chapin, W. Herndon, L. Notargiacomo, M. Katz, and T. Mowbray. Security for the Common Object Request Broker Architecture (CORBA). In *Proceedings 10th Annual Computer Security Applications Conference (Cat. No.94TH8032)*, pages 21–30. IEEE Comput. Soc. Press, Los Alamitos, CA, USA, 1994. ISSN/ISBN 0 8186 6795 8.

[CP96] G. Cattaneo and G. Persiano. Design and Implementation of a Transparent Cryptographic File System for Unix. Technical report, Dip. Informatica ed Appl., Universta di Salerno, Baronissi(SA), Italy, 1996. URL: ftp://edu-gw.dia.unisa.it/pub/tcfs/tcfs.ps.gz.

[CPS95] B. Callaghan, B. Pawlowski, and P. Staubach. NFS Version 3 Protocol Specification. RFC 1813, June 1995.

[CS92] Silvana Castano and Pierangela Samarati. An Object-Oriented Security Model for Office Environments. In Larry D. Sanson, editor, *International Carnahan Conference on Security Technology 26*, pages 146–152. IEEE, 1992.

[dBB94] B. den Boer and A. Bosselaers. Collisions for the compression function of MD5. In *Advances in Cryptology: Eurocrypt '93*, pages 293–304. Springer-Verlag, 1994.

[DH76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT(I22):644–654, Nov 1976.

[DHTK93] S. A. Demurijian, M.-Y. Hu, T. C. Ting, and D. Kleinman. Towards an Authorization Mechanism for User-Role Based Security in an Object-Oriented Design Model. In *Twelfth Annual International Phoenix Conference on Computers And Communications*, pages 195–202, New York, NY, USA, 1993. IEEE. ISBN 0-7803-0922-7.

[Dob96] Hans Dobbertin. Cryptanalysis of MD4. In *Fast Software Encryption: Third International Workshop, Cambridge, UK*, pages 53–69. Springer-Verlag, 1996.

[DOD85] US Department of Defense. Trusted Computer System Evaluation Criteria. DOD5200.28-STD, Dec 1985.

[ECM96] ECMA. Standard ECMA–219, Authentication and Privilege Attribute Security Application with Related Key Distribution Functions. 114 Rue du Rhône, CH-1204 Geneva, Switzerland, March 1996. http://www.ecma.ch/stand/ecma-219.HTM.

[GDS97] Mats Gustafsson, Benoit Deligny, and Nahid Shahmehri. Using NFS to implement Role-Based Access Control. In *Proceedings Sixth IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 299–304, MIT, Cambridge, Massachusetts, USA., June 1997. IEEE Computer Society Press. ISBN 0-8186-7967-0.

[GS96] Mats Gustafsson and Nahid Shahmehri. A Role Description Framework and its Applications to Role-Based Access Control. In *NORDSEC '96 – Nordic Workshop on Secure Computer Systems*, Gothenburg, Sweden, Nov 1996. SIG Security/Dept. of Computer Engineering, Chalmers University of Technology. Also presented at the IEEE WET ICE '96 Workshop on Enterprise Security, Stanford University, Palo Alto, June 19–21 1996.

[ISO96] Information technology — Open Systems Interconnection — Security frameworks for open systems: Access control framework. International standard ISO/IEC 10181-3, Sep 1996.

[JH92] K.M. Jackson and J. Hruska, editors. *Computer Security Reference Book*. Butterworht-Heinemann, Oxford, 1992. ISBN 0-7506-0357-7.

[Kle86] S. R. Kleiman. Vnodes: An Architecture for Multiple File Systems Types in Sun UNIX. In *Proceedings of the Summer 1986 USENIX Conference*, pages 238–247, Atlanta, Jun 1986.

[KN93] J. Kohl and C. Neuman. The Kerberos Network Authentication Service (V5) . RFC 1510, September 1993.

[Lin] The Linux Home Page. World Wide Web page. URL: http://www.linux.org/.

[Lin93] J. Linn. Generic Security Service Application Program Interface. RFC 1508, Sept 1993.

[Lin97] J. Linn. Generic Security Service Application Program Interface, Version 2. RFC 2078, January 1997. Obsoletes RFC 1508.

[LM91] X. Lai and J. L. Massey. A proposal for a new block encryption standard. In *Advances in Cryptology — Eurocrypt '90*. Springer-Verlag, 1991.

[LMSY95] Emil C. Lupu, Damian A. Marriott, Morris S. Sloman, and Nicholas Yialelis. A Policy Based Role Framework for Access Control. In *First ACM Workshop on Role-Based Access Control*, Gaithersburg, Maryland, Nov 1995. URL ftp://dse.doc.ic.ac.uk/dse-papers/management/rbac95.ps.Z.

[Mar93] Damian A. Marriott. Management Policy Specification. Research Report DoC 94/1, Imperial College, Department of Computing, 180 Queens's Gate, London, SW7 2BZ, Uk, Nov 1993. URL ftp://dse.doc.ic.ac.uk/dse-papers/management/policy_spec.ps.gz.

[McL87] William T. McLeod, editor. *The New Collins Concise Dictionary of the English Language*. Guild Publishing London, 1987.

[MD94] Imtiaz Mohammed and David M. Dilts. Design for Dynamic User-Role-Based Security. *Computers & Security*, 13(8):661–71, 1994.

[Mic] Sun Microsystems. Solstice Network Client – PC-NFS™ 5.1. Product information, World Wide Web document. http://www.sun.com/sunsoft/solstice/Networking-products/PC-NFS51.html.

[Mos92] Robin Moses. *Computer Security Reference Book*, chapter on Risk Analysis and Management, pages 227–263. Butterworht-Heinemann, Oxford, 1992. ISBN 0-7506-0357-7.

[Nat93] National Institute of Standards and Technology (NIST). Data Encryption Standard (DES). Federal Information Processing Standards Publication (FIPS PUB) 46-2, December 1993. http://csrc.ncsl.nist.gov/fips.

[Nat95] National Institute of Standards and Technology (NIST). Secure Hash Standard. Federal Information Processing Standards Publication (FIPS PUB) 180-1, April 1995. http://csrc.ncsl.nist.gov/fips.

[NO93a] Matunda Nyanchama and Sylvia Osborn. Role-Based Security, Object Oriented Databases & Separation of Duty. *SIGMOD RECORD*, 22(4):45–51, Dec 1993.

[NO93b] Matunda Nyanchama and Sylvia Osborn. Role-Based Security: Pros, Cons & Some Research Directions. *ACM SIGSAC Review*, 2(2):11–17, June 1993. URL http://www.csd.uwo.ca/tech-reports/442/roles.ps.Z.

[NO94] Matunda Nyanchama and Sylvia Osborn. Access Rights Administration in Role-Based Security Systems. In J. Biskup, M. Morgenstern, and C. E. Landwehr, editors, *Database security, VIII*, pages 37–56. IFIP, North-Holland, 1994.

[Nov] Novell, Inc. IntranetWare NFS ServicesIntranetWare and NetWare 4 Edition: Executive Summary. World Wide Web document. http://www.novell.com/catalog/qr/sne34210.html.

[NT94] B. Clifford Neuman and Theodore Ts'o. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications Magazine*, 32(9):33–38, September 1994. http://nii.isi.edu/publications/kerberos-neuman-tso.html.

[Opp96] Rolf Oppliger. *Authentication Systems for Secure Networks.* Artech House, 1996. ISBN 0-89006-510-1.

[Pfi96] Birgit Pfitzmann. *Digital Signature Schemes – General Framework and Fail-Stop Signatures.* Lecture Notes in Computer Science 1100. Springer-Verlag, 1996. ISBN 3-540-61517-2.

[Pfl97] Charles P. Pfleeger. *Security in Computing.* Prentice Hall, second edition, 1997. ISBN 0-13-185794-0.

[PJS+94] Brian Pawlowski, Chet Juszczak, Peter Staubach, Carl Smith, Diane Lebel, and David Hitz. NFS Version 3 Design and Implementation. In *Proceedings Summer 1994 USENIX Conference*, pages 137–151, Boston, MA, USA, 1994.

[PP95] Tom Parker and Denis Pinkas. SESAME V4 – OVERVIEW. World Wide Web document, Dec 1995. URL: http://www.esat.kuleuven.ac.be/cosic/sesame/doc-ps.html.

[PS95] Tom Parker and Chris Sundt. Role Based Access Control in Real Systems. In *First ACM Workshop on Role-Based Access Control*, Gaithersburg, Maryland, Nov 1995.

[Rap97] Karin Rappe. Roles and Role Management in Role-Based Access Control – Model, design and implementation. Master's thesis, Linköping University, Dept. of Computer and Information Science, Linköping University, S-581 83 Linköping, Sweden, Jan 1997. LiTH-IDA-Ex-9652.

[RC93] P. Rogaway and D. Coppersmith. A Software-Optimized Encryption Algorithm. In *First Workshop on Fast Software Encryption*, pages 56–63, Cambridge, UK, 1993. Springer-Verlag.

[Riv91] Ronald L. Rivest. The MD4 Message Digest Algorithm. In *Advances in Cryptology — CRYPTO '90*, pages 303–311. Springer-Verlag, 1991. Also Internet RFC 1320, April 1992.

[Riv92] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1320, April 1992.

[Riv95] Ronald L. Rivest. The RC5 encryption algorithm. *CryptoBytes*, 1(1):9–11, 1995.

[RL96] Ronald L. Rivest and Butler Lampson. SDSI - A Simple Distributed Security Infrastructure. URL, 1996. http://theory.lcs.mit.edu/~rivest/sdsi10.html.

[RSA78] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[Sal90] Arto Salomaa. *Public-Key Cryptography*, volume 23 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1990. ISBN 3-540-52831-8.

[Sal96] Arto Salomaa. *Public-Key Cryptography*, volume 23 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1996. Second enl. edition. ISBN 3-540-61356-0.

[SBS+95] Mark Shand, Donald Becker, Rick Sladkey, Orest Zborowski, Fred van Kempen, and Olaf Kirch. The LINUX User-Space NFS Server(1), Version 2.2. URL ftp://ftp.mathematik.th-darmstadt.de/pub/linux/okir/, December 1995.

[SCFY94] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-Based Access Control: A Multi-Dimensional View. In *Proceedings of the 10th Annual Computer Security Applications Conference*, Orlando, Florida, Dec 5–9 1994. URL http://www.list.gmu.edu/confrnc/acsac/ps_ver/a94rabca.ps.

[SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *Computer*, 29(2):38–47, Feb 1996.

[Sch95] Bruce Schneier. *Applied Cryptography*. Wiley & Sons, 1995. ISBN 0-471-11709-9.

[SF94] Ravi S. Sandhu and Hal Feinstein. A Three Tier Architecure for Role-Based Access Control. In *Proceedings of the 17th NIST-NCSC National Computer Security Conference*, pages 138–149, Baltimore, MD, October 11-14 1994. URL http://www.list.gmu.edu/confrnc/ncsc/ps_ver/b94rbac.ps.

[Slo94] Morris Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 2(4), 1994. URL ftp://dse.doc.ic.ac.uk/dse-papers/management/pdman.ps.Z.

[Sta94] William Stallings. SHA: The Secure Hash Algorithm. *Dr. Dobb's Journal*, pages 32,34, April 1994.

[Sun87] Sun Microsystems, Inc. XDR: External Data Representation Standard. RFC 1014, June 1987.

[Sun88] Sun Microsystems, Inc. RPC: Remote Procedure Call Protocol Specification Version 2. RFC 1057, June 1988.

[Sun89] Sun Microsystems, Inc. NFS: Network File System Protocol Specification. RFC 1094, March 1989.

[Tat97] Tatu Ylönen. Ssh (Secure Shell) Home page. URL, 1997. http://www.cs.hut.fi/ssh/.

[The95] The Object Management Group. *CORBAservices: Common Object Services Specification*, chapter Security Service Specification, pages 15–1 – 15–286. OMG, revised edition edition, March 1995. Updated 1996, 1997. URL: http://www.omg.org/.

[URLa] SPKI Certificate Documentation. URL. http://www.clark.net/pub/cme/html/spki.html.

[URLb] X.500 directory services — documentation. URL. http://www.bath.ac.uk/~ccsap/Directory/documentation.html.

[WR92] C. Weider and J. Reynolds. Executive Introduction to Directory Services Using the X.500 Protocol. RFC 1308, March 1992.

[WRH92] C. Weider, J. Reynolds, and S. Heker. Technical Overview of Directory Services Using the X.500 Protocol. RFC 1309, March 1992.

[Yag96] T. Yager. Linux matters. *BYTE*, 21(2):123–4,126–8, Feb 1996.