LEABHARLANN CHOLÁISTE NA TRÍONÓIDE, BAILE ÁTHA CLIATH | TRINITY COLLEGE LIBRARY DUBLIN
Ollscoil Átha Cliath | The University of Dublin

Terms and Conditions of Use of Digitised Theses from Trinity College Library Dublin

Copyright statement

All material supplied by Trinity College Library is protected by copyright (under the Copyright and Related Rights Act, 2000 as amended) and other relevant Intellectual Property Rights. By accessing and using a Digitised Thesis from Trinity College Library you acknowledge that all Intellectual Property Rights in any Works supplied are the sole and exclusive property of the copyright and/or other IPR holder. Specific copyright holders may not be explicitly identified. Use of materials from other sources within a thesis should not be construed as a claim over them.

A non-exclusive, non-transferable licence is hereby granted to those using or reproducing, in whole or in part, the material for valid purposes, providing the copyright owners are acknowledged using the normal conventions. Where specific permission to use material is required, this is identified and such permission must be sought from the copyright holder or agency cited.

Liability statement

By using a Digitised Thesis, I accept that Trinity College Dublin bears no legal responsibility for the accuracy, legality or comprehensiveness of materials contained within the thesis, and that Trinity College Dublin accepts no liability for indirect, consequential, or incidental, damages or losses arising from use of the thesis for whatever reason. Information located in a thesis may be subject to specific use constraints, details of which may not be explicitly described. It is the responsibility of potential and actual users to be aware of such constraints and to abide by them. By making use of material from a digitised thesis, you accept these copyright and disclaimer provisions. Where it is brought to the attention of Trinity College Library that there may be a breach of copyright or other restraint, it is the policy to withdraw or take down access to a thesis while the issue is being resolved.

Access Agreement

By using a Digitised Thesis from Trinity College Library you are bound by the following Terms & Conditions. Please read them carefully.

I have read and I understand the following statement: All material supplied via a Digitised Thesis from Trinity College Library is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of a thesis is not permitted, except that material may be duplicated by you for your research use or for educational purposes in electronic or print form providing the copyright owners are acknowledged using the normal conventions. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone. This copy has been supplied on the understanding that it is copyright material and that no quotation from the thesis may be published without proper acknowledgement.

A Unified Access Control Mechanism

Joerg Abendroth

A thesis submitted to the University of Dublin, Trinity College

in fulfillment of the requirements for the degree of

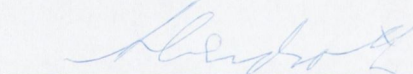
Doctor of Philosophy (Computer Science)

September 2004

TRINITY COLLEGE
0 5 AUG 2005
LIBRARY DUBLIN
THESIS
7700

Declaration

I, the undersigned, declare that this work has not previously been submitted to this or any other University, and that unless otherwise stated, it is entirely my own work.

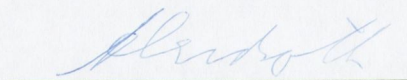


Joerg Abendroth

Dated: 20 September 2004

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.



Joerg Abendroth

Dated: 20 September 2004

Acknowledgements

'*A man is always moving either forward or backward,*'
says Kano, 9-dan,'*He never stands still.*'[1]

Kano is a professional Go player. In Asia being a professional Go player is as much respected as having a PhD. The path to achieving that goal comprises also many similarities. For one it often starts with the help of one's parents - I am very grateful for the everything my parents did for me ! Other similarities include determination and ambition as well as skill and luck. Both paths become easier if one has a sensei; much like the a supervisor. A good sensei not only helps one by teaching factual knowledge, but even more by leading the direction and setting the right set of goals. In some cases a sensei even takes the young Go player into his family and sponsors him financially. For my PhD studies, one might say I had two sensei Christian Jensen (the spiritual ;) and Vinny Cahill (a bit more earthly). I have much gratitude for both.

But to become a professional Go player, other factors are equally important. It is known that improving ones play is easiest by playing people equal and stronger than oneself. This translates in academia to having a good research group , and I had the luck to have had two. I like to thank not only the Distributed Systems Group, but also the BRICS Århus who provided me with a inspiring working environment, set chal-

[1]Taken from *Lessons in the Fundamentals of Go*, Kiseido Publishing Company, Tokyo, 1978

lenges and simply provided (mental) sparring partners for discussions which elevated the level of my studies. Special thanks have to go to Jean Marc Seigneur, Mogens Nielsen, Gregor Gaertner, Hans Huettel, Jens David, Pawel Sobinski, Claas Roever, Colin Harris and Arthur Carter.

The proverb says that 'A man is always moving either forward or backward'. Indeed, a backward movement on the emotional side of a person can affect the level of professional play. Equally I had during my PhD studies some ups and downs and I am grateful for all the non academical friends I have, no matter how short or long they joined me on my way. I like to thank Thomas Knape, Jeanette Lever, Barry Callagher, Silke Wolter, Andreas Grab, Juan Franscisco Golenowski, Peter Brouwer, John Gibson, Marcus Schirmer, Jagoda Espensen, Meike Hartmann, the AIESECers, the whole of the Pembrokeians and last but not least all the different housemates I had.

Finally the proverb says 'He never stands still', so I am looking forward to the time ahead and hope I can be as good a friend to all of you, as you were to me. Many thanks once again.

Joerg Abendroth
University of Dublin, Trinity College
September 2004

Abstract

The widespread employment of the Internet facilitates collaboration and the use of distributed applications. Security concerns, such as controlled access to resources are preeminent. Often different applications employ different access control models, which are extended by each organisation or administration domain to enforce their own policies. Maintaining a coherent and non-complex access control policy becomes an increasingly difficult task.

An important building block in a coherent access control policy is to be able to utilise the same policies in all employed applications. Application development today views access control as a hybrid component consisting of access control model or policy, and an access control enforcement mechanism. To be able to employ the same policy across different applications, or to employ different policies in the same application, depending on context, requires that the access control enforcement mechanism is separated out.

In this thesis we define a unified access control framework for distributed applications called ASCap[2]. The design of the ASCap framework is based on an analysis of different access control models, which identifies their key requirements and helps define the different components of the presented access control framework. The ASCap framework is able to instantiate various access control models using the same underlying access control mechanism. This gives companies employing the ASCap framework

[2]Active Software Capabilities

the freedom to use exactly that access control model most suitable for their purpose and allows migration of access control models whenever needed.

The ASCap framework is founded on the ideas of proxy-based access control and active capabilities, which are extended to suit the setting of open distributed systems. The ASCap framework uses external security servers to increase flexibility, up to the point where access control models can be decided on a per user (group) basis, ultimately adapting the access control to the business model.

ASCap is the first access control framework to allow partial outsourcing to jointly administrate a security domain in a controlled manner. Some benefits of the new paradigm of partial outsourcing will be demonstrated by combining access control and intrusion detection. Using partial outsourcing new *active* intrusion detection strategies are proposed. These strategies allow systems to probe for intruders that do not produce other visible alarm signs.

The ASCap framework provides a unified client and server interface without altering the behaviour of the employed access control model. A formal model using π-calculus has been employed to show that the access control model behaviour is not affected. This proof of flexibility itself presents a contribution to the formal methods research body.

In this thesis we have developed a prototype implementation of the ASCap framework in order to provide basic proof-of-concept, experiment with several well-known security models and demonstrate how they are instantiated within the ASCap framework and finally to allow empirical performance overhead estimation.

Publications Related to this Ph.D.

[1] J. Abendroth and C. D. Jensen *A Unified Security Framework for Distributed Applications.* Proceedings of 18th ACM Symposium on Applied Computing, March 2003.

[2] J. Abendroth and C. D. Jensen *Partial Outsourcing: A New Paradigm for Access Control.* Proceedings of 8th ACM Symposium on Access Control Models and Technologies, June 2003.

[3] J. Abendroth *Applying π-Calculus to Practice: An Example of a Unified Security Mechanism.* BRICS Report Series RS-03-39, November 2003.

[4] J. Abendroth *Aktive Strategien zur Schutzzielverletzungerkennung durch eine kontrollierte Machtteilung in der Zugriffskontrollarchitektur (Achieving Active Strategies for Intrusion Detection by Using Partial Outsourcing.)* GI Workshop on Detection of Intrusions and Malware & Vulnerability Assessment, July 2004.

Contents

[3]A version of this section was published in an earlier paper [4].

[4]A version of this section was published in an earlier paper [5].

[5] A version of this section was published in an earlier paper [3]
[6] A version of this section was published in an earlier paper [4].

List of Tables

List of Figures

Abbreviations

ACP	Access Control Programs
ACL	Access Control List
ACS	Access Control Server
AES	Advanced Encryption Standard
AI	Artificial Intelliegence
ASCap	Active Software Capabilities
ASL	Authorisation Specification Language
ATAM	Augmented Typed Access Matrix
BLP	Bell La-Padula (Access Control Model)
CA	Certification Authority
CCS	Calculus for Comunication Systems
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
DAC	Discretionary Access Control
DES	Data Encryption Standard
DRBAC	Distributed Role Based Access Control
DTAM	Dynamic Typed Access Matrix
GRID	Global Resource Information Database
HRU	Harrison Ruzzo Ulman
ICAP	Identity Based CAPabilities

IDL	Interface Description Language
IDS	Intrusion Detection System
IPC	Inter Process Communication
IPS	Intrusion Prevention Systems
IT	Information Technology
J2EE	Java 2 Enterprise Edition
JINI	Java Infrastructure Initiative
LDAP	Light Directory Access Protocol
LOGOS	Light Out Ground Operation System
MAC	Mandatory Access Control
MWB	Mobility Workbench
NIST	National Institute of Standards and Technology
OASIS	Open Architecture for Secure Interworking Services
ORB	Object Request Broker
PC	Personal Computer
PICT	Pi Calculus Typed (programming language)
PKI	Public Key Infrastructure
RAM	Random Access Memory
RBAC	Role Based Access Control
SMM	Subject Managment Module
SMS	Short Message Service
SPI	Secure PI Calculus
SPL	Secure Policy Language
SSL	Secure Socket Layer
SSO	System Securty Officer
TAM	Typed Access Matrix

TCB	Trusted Computing Base
TCPA	Trusted Cumputing Platform Alliance
XML	Extensible Markup Language
WAN	Wide Area Network

Chapter 1

Introduction

Businesses today differ from those of a pre-Internet era. Not only has e-commerce introduced a new type of business [124, 58], but structure, marketing and entire business processes have changed dramatically. The business structure is moving away from large conglomerate companies towards interdependent specialised companies [144]. Business processes such as supply chain management require that IT-systems of different companies are interconnected and synchronised in order to guarantee timely and automated delivery [155]. This requires IT security solutions which can be integrated with each other and adaptive to a dynamically changing environment. Setting up a new business relationship often requires some level of integration of the customer's and the supplier's IT systems and in particular their security policies. As business relationships may change spontaneously, security policies need to adapt dynamically.

With respect to security, the company intranet is no longer similar to a medeival fortress with one clearly defined gate that allows communication with the outside world (the company firewall normally takes the role as the gate to the outside world), but must be regarded as an open city with individual outward connections from different departments inside the company intranet. Much access control research is based on

the assumption that one single large TCB[1], and hence access control model, exists. This is no longer safe, because today's off-the-shelf software often implements its own access control mechanisms that customers are forced to use. This denies companies the freedom of defining their own security policy and maintaining it across the company and its software applications, because the specific access control mechanism may be unable to support the company policy. An access control framework should therefore allow free choice of security policies by providing a unified interface to different underlying access control mechanisms.

Assuming that the access control policy can be freely chosen, the next step would be to allow domain experts to formulate parts of the access control policy. This is an important motivation for existing research in the area of access control policies. One benefit is that each part of the policy will be defined by the person most knowledgeable in that domain. Renowned security expert Bruce Schneier summarises this with [147, 8]:

> Modern society is built around specialization; more tasks are outsourced today then ever before. We outsource fire and police services, government (that's what a representative democracy is), and food preparation. Businesses commonly outsource tax preparation, payroll, and cleaning services. Companies also outsource security: all buildings hire another company to put guards in their lobbies, and every bank hires another company to drive its money around town.

He goes on to say that for IT security:

> Companies should outsource expert assistance: vulnerability scanning, monitoring, consulting, and forensics, for example. But they should not outsource management.

[1]Trusted Computing Base

This underlines the point, that it has to be possible to outsource aspects of access control to specialist companies while at the same time retaining full control of one's own network administration. In this thesis, we investigated how access control can be outsourced in a controlled manner. We propose partial outsourcing as the way to allow external expertise companies to add value to the security of IT systems.

One specific area where a company may benefit from external knowledge is in intrusion detection systems (IDS). Two major strategies have evolved in the area of intrusion detection. One utilises expert systems, which are trained on a normal system state and raise an alarm if the network traffic differs from the one seen in the training period (these systems are called anomaly detection systems). The other strategy aims to learn signatures of known network attacks and to install sensors in the network to detect these profiles. External companies are assumed to provide the IDS sensors with updates of the signature databases (this is very similar to the way most anti-virus software works). There have been attempts to enhance this passive scanning of network traffic with active elements. The resulting systems are called intrusion prevention systems (IPS) and are able to actively modify network traffic in the event of an intrusion alarm. However, we claim that IPS also devise only passive intruder recognition techniques by monitoring the network traffic. We propose a new *active* probing strategy which allows the system to change the access control framework to make hidden intruders visible. During the active probing, the environment is altered to provoke alarm signs from the intruder, which otherwise would stay hidden. An example of active probing can be the introduction of an additional authentication protocol like SMS verification. In a normal state, such an authenticaiton would be to cumbersome, but as a tool for active probing it is warrantable.

A review of recent programming suites shows that all aim to facilitate the building of interdependent applications: CORBA [126] provides a message passing facility with location transparency, J2EE [98] allows applications to integrate by the use of mobile

proxy code, and web services [166] provide a front-end to integrate different business applications using the Worldwide Web. These technologies promote component-based models to allow the building of applications based on those components that are relevant to the company's needs and infrastructure, including security related components. A strong motivation for the design of a new access control mechanism is to facilitate the integration of not only different application components, but also security policies.

Our work also has been inspired by existing work in the area of access control model research. Various models have been proposed and employed. Discretionary access control models, e.g. based on access control lists (ACLs), are common in existing applications; while mandatory models, e.g. multilevel security models, exist, for example, in the military world. New models have been proposed to facilitate the task of administrating large numbers of users in diverse organisations. Role-based access control has found acceptance and is integrated into business networks [88]. A desired scenario allows an organisation to employ the security model which best fits the environment of the deployed applications. Obviously, different security models will require a different infrastructure which results in the need for a unifying access control mechanism to provide a standardised security component interface.

One area of research, which aims to integrate different access control models (and general IT policies), is access control policy research. Policy research has an emphasis on the unambigous description of all aspects of access control models and assumes a functioning policy deployment infrastructure. In some cases the existing environment will not be able to provide the deployment infrastructure required, so it is worthwhile to examine access control specification from the point of view of the access control mechanism. This means that we view the access control models and policy languages as two kinds of upper layers, which have to be supported by an underlying infrastructure.

Research in access control mechanisms is, in most cases, limited to the fitness for a certain environment. For example, the design goal of Kerberos was to provide au-

thentication in open network systems; and AMOEBA provided capability based access control for a multi-node distributed operating system. In this thesis we extend this research to include support for various access control models, thereby facilitating the deployment of the mechanism in a wide range of scnearios. Naturally, some environments will prohibit the correct implementation of some access control models, hence some of our research included reasoning about these interdependencies. For the benefit of security system architects we derived the security-flexibility-performance principle, which can be used as a guideline for practical systems design.

1.1 Thesis

This thesis addresses the problems of providing access control in distributed applications across different administration boundaries. Such an access control framework for distributed applications has to fulfill requirements in different independent areas, which were presented above and can be summarised as follows:

- allow free choice of access control model and security policy
- allow partial outsourcing of access control properties and management
- allow dynamic changes of security policy
- provide a unified interface to be employed in different applications
- provide scalability to a large number of users
- do not add an overly expensive overhead

Within the context of this thesis we will design and implement an access control framework (ASCap[2] framework) which meets all these requirements. This is done

[2]Active Software Capabilities

by combining proxy-based access control [117], active capabilities [131] and the introduction of an external security server approach. The proxy principle provides the application with a standardised security interface, while being able to adapt both the client-side and server-side security component. Active capabilities, as well as hidden software capabilities, allow a flexible access control mechanism approach, as opposed to a static mechanism which requires flexible policies. Analysis of related research and access control models showed that in some cases an extended trusted computing base is needed. This can be achieved through the use of an external security server which could, for example, be a smartcard device connected to the local PC.

1.2 Contributions

The ASCap framework provides a unified access control mechanism for various access control models. The evaluation of the ASCap framework uses examples of different access control model setups, as well as a formal behavioural model written in π-calculus. Finally the possibilities of partial outsourcing are shown by example applications which have been implemented.

The primary contributions of this thesis are:

- The novel external security server approach adds to the flexibility of the framework and allows it to emulate Kerberos-like access control frameworks. This is the first major contribution of this thesis. (Chapter 4)

- Partial outsourcing: a new paradigm for access control, can be seen as the second major contribution. We identify different categories of outsourcing access control (Section 5.8) and show how, in the area of IDS, new detection strategies become available by the use of partial outsourcing. (Section 7.5)

- The novel use of π-calculus to show that the ASCap framework does not influ-

ence the instantiated access control model is a significant contribution to applied computer science. (Section 7.4)

- The extension and combination of the active capability and hidden software capability approach to form the active software capabilities. Employing the active software capabilities according to the proxy-principle allows a substantial increase in flexibility. (Chapter 4)

1.3 Road Map

The remainder of the thesis is structured as follows:

Chapter 2 presents the area of computer security and the environment of open distributed systems. This chapter provides the foundation for the rest of the thesis by presenting the underlying assumptions of the proposed framework.

Chapter 3 This chapter examines different fundamental access control models and access control policy approaches. Each of these is reviewed with respect to the requirements that are needed from the underlying access control mechanism. Access control mechanisms are then reviewed under the same criteria judging the fitness of instantiating access control models.

Chapter 4 reviews the past research results, which have influenced the design of the ASCap framework. While giving the definitions of each of the core elements, the original concepts will be related to our design, by highlighting similarities and reasons behind modifications. This chapter will be concluded by a discussion of interdependencies in practical security system designs.

Chapter 5 presents the design of the ASCap framework and describes the relevant components under different design aspects. A security analysis also discusses

how delegation or revocation can be handled. The final section categorises access control outsourcing and names the new paradigm of access control.

Chapter 6 presents a prototypical implementation of the ASCap framework.

Chapter 7 evaluates the ASCap framework using the strategies outlined at the beginning of the chapter. The evaluation encompass all criteria named in the review sections of the thesis. A special emphasis is on the formal evaluation of the framework's flexibility, which is done by a novel use of the π-calculus. The evaluation concludes by a preview of additional benefits of the ASCap framework in the area of intrusion detection.

Chapter 8 presents the conclusions of this thesis and discusses future work.

Appendix A provides the formal basics of the π-calculus used in Section 7.4.

1.4 Summary

This chapter introduces trends in today's business IT infrastructure and relates them to the area of access control. These build the motivation for the work in this thesis. An outline of the following chapters was also given.

Chapter 2

Security and Distributed Systems

The term security is always connected with protection. Protection raises the questions of: What? Against whom or what? How? The first 'what' concerns the assets, which have to be protected. 'Against whom ?' usually leads to a person, an attacker, but it might be something else. Ideally one would want to name all possible attacks, categorise them and develop countermeasures for each attack. Existing research investigates 'what' kind of security breaches exist and how they can be categorised [65, 103, 28]. Defining the term security in each of these categories will lead to contradictions.

Bruce Schneier writes in his book "Secrets & Lies" [148] about security, protection and security measures. He stresses the importance of understanding not only the security measures but also the risk relationships of these. He gives the example of a safe, which should prevent an attacker from stealing the valuables in it. A safe does not stop an attacker with the right set of tools and a fair amount of time; all safes can be broken into. The task of choosing the right safe therefore involves analysing the surrounding environment and security measures, such as a guard service. Then the right safe only needs to provide protection until the next guard tour arrives. Equally in all other areas of security, one protective measure does not work on its own, but provides some properties other measures rely on. Another important fact that must be

noted from this example is the relationship of risk and cost of protection. A company will not install a safe and other protective measures, which cost twice as much as the valuables protected by it.

In the remainder of the chapter we will discuss general security and protection requirements (Section 2.1). Section 2.2 introduces different approaches to information access security and provides a general introduction to the next chapter on access control security, by relating the two branches access control model and information flow model research. Then Section 2.3 provides an introduction to the area of network security. The following Section 2.4 does the same for the area of distributed systems security. Finally Section 2.5 summarises the content of this chapter.

2.1 Computer Security

In the area of computer security, the assets are typically information. Information includes many things: such as information about real world valuables: the combination to the safe, construction plans, or customer account details. A company may provide a service using information, such as a telephone directory, in which case it is not only vital that the information is correct, but also readily available. A list of protection requirements relating to these information assets one often finds in security literature is ([47]):

- Confidentiality
- Integrity
- Availability

Other lists include *authenticity*, *privacy* and *accountability* depending on the background and time period. We will discuss the first three requirements and add additional requirements at various points in this thesis.

Confidentiality: Confidentiality demonstrates the close relationship between security and secrecy. Information should not be disclosed, and sometimes not even known about, to outsiders. Confidentiality can be achieved by combining different measures, such as cryptography, steganography or access control. Many of security requirements are rooted in the need for confidentiality.

Integrity: For information to be valuable it needs to be correct, and that includes completeness up to the permissable level of expense. Integrity protection measures therefore deal with modification of the data. An offer to sell a certain merchandise may be known by everybody, but it is important that the price cannot be changed if the offer is to be legally binding. In the electronic commerce such an offer may be cryptographically signed or be displayed in a 'read-only' location such as the company's web site. It is claimed that integrity is closely related to *accountability*, because if one is able to deny the origin of one's offer its integrity becomes void.

Availability: Availability is a security requirement if the business model consists of generating extra revenue with information that may already be in the public domain. A company, which provides a user-friendly telephone number search engine may charge for this service, although everybody has their own phonebook at home. In this case the availability of information becomes important. From the three presented requirements availability is the hardest to protect. Predicting the requirements on the computer system to provide the information under normal conditions is feasible. However an artificially high demand such as during a distributed denial of service attack [103], may be much larger than predicted and unmanageable for the implemented system. Denial of service attacks can be hard to counter, because each request alone can be fully valid.

2.2 Access Decision and Information Flow

The security aims presented in the previous section all require some sort of access government. Confidentiality can be guaranteed only if authorised personnel are permitted to read the information. Integrity is protected by prohibiting malicious users from modifying the information. Each access request has to be evaluated. We call this function the access decision function.

2.2.1 Access Decision Function, Policy and Models

The access decision function evaluates, for a certain subject, the right to execute an operation on a certain object in a specific system state. In early systems, a central reference monitor implemented the access decision function directly [7]. Current designs are based on an abstraction of three layers, which is shown in Figure 2.1. The access decision implements a specific access control policy of a specific access control model.

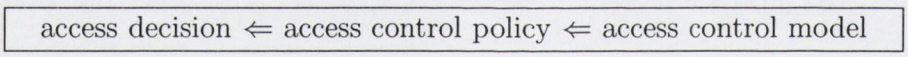


Fig. 2.1: Abstractions of Access Control Security

An access decision gets all required input parameters and outputs a simple 'access granted' or 'access denied' decision. The access decision is done according to the system's policies. System policies describe, for example, (1) desired reaction to events, (2) that certain system states should not be reached, or (3) how to derive access rights in certain system states. A system policy is defined by the local system administrator. These policies may not only be limited to access control, but include maintenance or accounting tasks as well. Generally, access control policies will be written according to access control models. An access control model may be predominantly concerned with confidentiality (like the Bell-Lapadula Model [14]) or integrity (like the Clark-Wilson model [27]). In all cases the access control model will give direction as to which kind of

access control policies an administrator should use and ultimately influences each single access decision. We will review the past access control research in the next chapter 3.

2.2.2 Information Flow Model

A different approach is taken by the information flow model. Information flow research was spawned from early access control research, after Lampson had discovered the confinement problem [90] or the problem of covered-channels. Given an information system, concerned with confidentiality, a subject may be allowed to read top secret information but, to prevent this subject from disclosing information, it is prohibited from writing outside its own security group. Subjects at a public security level may still input information into the top secret group but, of course, are not allowed to read any. The system now faces a problem if a public subject attempts to write to an object which a top-secret subject has in use. An intuitive implementation may raise an resource occupied exception, which itself contains some kind of information. It has been noted that a low bitrate covered channel can be set up by such an access violation notification.

Information flow security research [34] aims to solve this problem. Different approaches have been proposed. A formal analysis can be either static or dynamic. A static analysis considers the system as a static object. All possible object interactions (and sequences thereof) have to be considered, only if no undesired information flow occurs in a system is it called secure. A dynamic analysis considers the system under execution. This will prevent some system states from occurring, which makes the dynamic information flow analysis less restrictive than the static. Typical information flow research projects include analytical compilers [11, 133, 109], which generate proof carrying certificates along executable code.

A related research area is *non-interference models*, which provide a formalism of

what each subject knows of the state of the system. "Subject s_1 does not interfere with subject s_2 if the actions of s_1 have no influence on s_2's view of the system." [47].

Because of the limited number of practical information flow implementations it is not clear what requirements such a security model puts onto an access control mechanism. Therefore, for the scope of this thesis, we restrict our research to the area of access control models and policy languages.

2.3 Network Security

Network security, like the term security itself, is not unambiguous. Network security includes:

- Sniffing of network traffic (confidentiality breach)
- Spoofing of network traffic (integrity breach)
- Denial of service attacks (availability)
- Intrusion attempts on network nodes (such as web server)
- Attacks on network boundaries (e.g. intrusion into an Intranet)
- Disclosure of identity of anonymous user

In single computer systems the first two attacks could be countered by access control measures. The nature of computer networks requires that all intermediate network nodes have to get access to the full information to deliver it to the final recipient. To prevent sniffing and spoofing attacks, cryptographic measures have to be used.

2.3.1 Sniffing and Spoofing of Network Traffic

Both forms of attack on network traffic can be prevented by the use of cryptography. There are three types of cryptographic systems.

1. Shared Key Systems

2. Public Key Systems

3. Quantum Cryptography Systems

It is only recently a commercial quantum cryptography system appeared on the open market. However these systems still include many limitations which prevent widespread use. Once quantum cryptography systems can be employed many of the key agreement and sharing problems will be solved. For example in shared key cryptography the sender and receiver both share the same key. The longer the key the more secure the system against computationally intensive attacks. An ideal key is a one-time pad, which provides a key bit for each clear text bit. One-time pads are theoretically information secure. The main problem of shared key cryptography is the exchange of the shared key.

The Diffie-Hellmann protocol can be used to establish a shared key without prior knowledge. However the Diffie-Hellmann protocol is prone to the man-in-the-middle attack, in which the attacker establishes two keys one each with the two parties of the Diffie-Hellmann protocol. To prevent the man-in-the-middle attack a timestamp and synchronised trusted clocks can be used.

Public key cryptography utilises so-called trapdoor functions to establish a secure communication channel. Encryption and decryption is done with a key pair. The public key can be publicly known and used by anyone to encrypt secret communication to the receiver. The private key, known only to the receiver, is the only key able to decrypt such encrypted information. Again the public key distribution faces an integrity problem, which can be solved using trusted third parties. Such parties are either certification authorities (in case of PKI [163]) or a chain of trusted key referrers (such as at the PGP's Web of Trust [73, 80]).

Cryptography has only recently been applied to other fields than communication security and secret multi-party computation. Harrington and Jensen [54] proposed a cryptographic access control system using keys as capability tokens.

2.3.2 Intrusion Attempts on Network Nodes

Computer networks not only provide information and services to distant physical locations, but also allow attackers to gain access at those remote nodes. Whether it is the physical distance or the abstract level of actions, intrusion attempts on network nodes account for the majority of network security incidents. A counter-measure could be to divide the network into security domains, which are guarded by boundary enforcement systems (firewalls).

2.3.3 Denial of Service Attacks

Denial of service attacks are the hardest to prevent. The task is to decide whether a correctly formatted and fully valid access request has actually been sent with the purpose of keeping the server busy in order to prevent it answering genuine access requests. Today distributed denial of service attacks employ a large number of (usually) hijacked personal computers to damage the business capability and reputation of large websites.

2.3.4 Attacks on Network Nodes

Attacks on network nodes exploit programming or design flaws in the employed software. In many cases buffer overflows allow attackers to inject arbitrary executable code into a remote server. Techniques to discover weaknesses in program code include verification techniques [67] and the use of automated source code checkers [158, 115].

2.3.5 Attacks on Network Boundaries

A wide range of firewall systems exist to protect vulnerable Intranet nodes. According to Schneier's principle of diversified measures [148] today's network boundaries not only guard access to network sections, but also monitor traffic to raise an alarm if (successful) intrusion attempts have been noted. Intrusion detection systems (IDS) can be divided into (I) passive intrusion detection and (II) active intrusion prevention systems. Passive systems monitor traffic for attack signatures and raise an alarm. Active systems are able to interfere with the network traffic and modify the communication stream. An active intrusion prevention systems may not be able to stop an intrusion attempt (due to the formerly unknown method of exploitation) but can prevent the well-known ways of access right exploitation, thus rendering a successful attack useless. We will introduce in this thesis a third class of active intrusion *detection* systems (see Section 7.5). The '*Active*' in this case relates to the strategy employed to discover intruders.

2.3.6 Disclosure of Identity of Anonymous User

Computer networks are used for a wide range of activities. In some cases anonymity is desired. For example a company monitoring my web surfing habits may be able to construct a personality profile of me. The use of this profile can lead to disadvantages (for example denial of insurance, because I view too many DIY sites). The correctness of these profiles and the right of anonymity are widely discussed [162]. From a technical point of view, attacks on user privacy are part of network security, while the main concern of this thesis is access control.

2.4 Distributed Systems Security

Distributing the workload across several computers has changed the way computer software is organised. In this section we will review two major influences in today's

distributed applications, namely (1) location and extent of the trusted computing base, and (2) flexibility of application interaction. Finally we will identify requirements for future open distributed systems with respect to access control frameworks, by means of a simple example scenarios.

2.4.1 Location and Extent of the Trusted Computing Base

Early computer systems were single mainframe computers operated by a government or large organisation. When it became possible for multiple users to use the same machine, time-sharing operation systems were developed [91, 82, 170]. From a security point of view a time-sharing system consists of trusted hardware with a trusted operating system acting as a reference monitor running untrusted user software. In effect each system had its own trusted computing base (TCB). Later operating systems research proposed distributed operating systems [49], which provide location and resource transparency. Some of these systems, such as shared memory systems employ a distributed infrastructure on each node. Besides the shared memory service these operating systems also provide security measures, which still includes a trusted computing system on each node. The Internet has become a collection of independent network nodes, which are connected via platform independent middleware (e.g. CORBA [126]). Application objects are able to call service objects with full location transparency, while no unified distributed OS is needed. In the case of CORBA a client is able to install its own object request broker (ORB), which interacts with other ORBs, to access services in the whole network, but no TCB is required on the client side. One difference of today's open distributed systems to early computer federations is that not all nodes provide a trusted computing base.

Figure 2.2 compares the extent of the TCB in the different development stages. The early time-sharing system consisted of one network node, which had different user programs running on top of a TCB. Shared memory systems used a special distributed

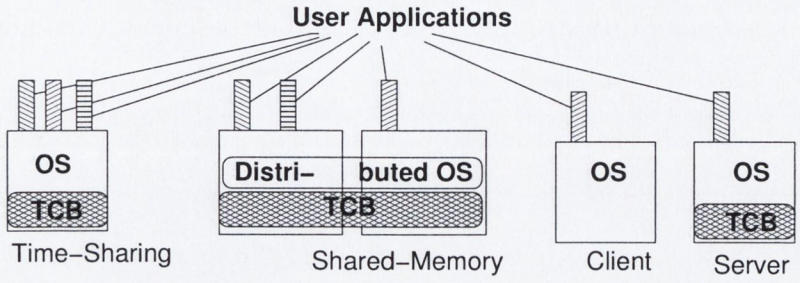


Fig. 2.2: Comparison of Location of TCBs at Different Systems.

operation system which provided, among other services, a TCB on each node. Modern distributed applications in today's Internet do not have a TCB on each node, specifically not on the client side.

2.4.2 Increased Flexibility

As the components of distributed applications change, the way applications interact changes too. Applications have to be able to bind new components into existing frameworks without requiring changes in the application code. Likewise a client contacting a new server may be required to act according to a new access control model - this requires great flexibility from the client. In order to provide this flexibility Bull et al. introduced the security design for open distributed processing [24]. Open distributed processing views each service and client as an object. Security requirements can be declared as part of the object using attributes declared in the interface. Hence the security policy of the object can change depending on the locality by changing the interface attributes. We adapt this notion of local policy dependency of security services and formulate the following as a requirement for an access control mechanism:

The access control model or policy has to be dynamically changeable.

This means that, depending on the context, a client is required to act according to different access control models. Dynamic changes also occur when the state of the

service provider needs to alter the access control model while maintaining the service connection with the client.

Bull [24] also notes that an object acting as a client to one service can be a service provider to another object. To be able to enforce for example mandatory access control models the authors assume that "the server can validate the service chain to the final client". This can be difficult in cases where the client does not cooperate and a central auditing authority does not exist. The solution proposed is identity-based capabilities (as also used in the ICAP [48] architecture). However, explicit delegations can be guarded using this approach, but implicit delegation such as a client acting as a proxy for a service cannot be detected. We will discuss a solution to this problem in Section 5.7.7.

2.4.3 Requirements of Open Distributed Systems

The environment in which the access control mechanism presented here will be employed is assumed to be similar to the open distributed processing design. Our point of view will not be limited to a purely object-oriented description, but will include elements that depict components of the Internet. For example, a client is an application running on an independent network node, and regarded to not be part of the TCB. A server is running on a protected platform and thus can be assumed to be inside the TCB. In a later section we will introduce different external servers, whose employment adds to the flexibility. External servers may also belong to independent administration domains which may require the server to protect itself from malicious input from these external security servers. Section 5.8 will show how this can be done.

In the following we will introduce each aspect of open distributed systems first in a general form and then with a concrete example. All examples will be from the setting

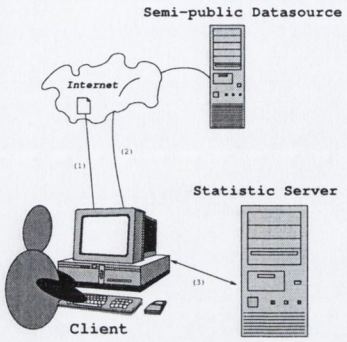


Fig. 2.3: Example Scenario of an Open Distributed System.

introduced in Figure 2.3. A professor is acting as an independent advisor for a financial institution which runs an application to provide her/him with information. There are different sources for the information, (1) public Internet pages, (2) the database of a non-profit organisation for his profession (e.g. ACM, IEEE), (3) a statistical program to evaluate trends (e.g. written and maintained by his department) and (4) the information system for the respective institution she/he is advising.

Client OS is not part of the TCB: In today's Internet the home PC participates in online business just like any other company computer. In a business-to-business relationship one can assume that the customer company's purpose for interaction is to facilitate the business process in its own sector. However the reason of some private persons may differ. In the case of a malicious customer, chances are high that he/she fully controls his own computer, allowing any sort of manipulation including re-routing of hardware connections to inject forged device replies. The conclusion is, that the client PC cannot be trusted - therefore the client OS is not part of the TCB.

In the setting of the university it is also often the case that staff are entrusted with maintenance of their own PC, which translates to the professor's computer being an independent untrustworthy node for most of the information sources.

Dynamic Access Control Model Change: Open distributed systems consist, by definition, of many independent data sources, each of them imposing its own access control model onto the client. A client, that wishes to interact with different sources during the course of one session, is required to adapt to the access control models dynamically.

Our university professor also has to adapt to different access control models. For example the database of the non-profit organisation currently employs an access control model based on the IP-address, but will be enhanced in the near future to a usage-oriented model. So his application may today only send the request to the database, while tomorrow it is required to send an account name.

No Central Authority: Open distributed systems, like the Internet, do not have a central authority, but have authorities for small security domains.

The professor gathers information across administration boundaries. Some may have administration trust-relationships with the client, such as the statistical program of the professor's department depicted (3). But the access control mechanism cannot assume that this is the case for all sources.

No Prior Knowledge of Data Sources: On the Internet new sites and services are offered every day. Existing companies convert to new formats, such as Google thinking about employing RSS feeds for their search engine. Many new services bring along advantages for the user, but require the client software to adapt to new data structures.

In our example, after a chat with a college, the professor might decide to join another news service, which provides not only charts, but also access to the raw data via a custom interface. An ideal case would allow him to connect the client application to this new service and incorporate its data as if it was provided by

the user manually. From the access control mechanism this requires to not only the ability to adapt to new access control models, but also different sources.

Performance optimisation: The popularity of certain web sites, such as music file sharing, have not only brought legal consequences but also shown that high performance requires a big technical effort. One solution is peer-to-peer (P2P) programs, which change the server-client relationship. P2P-servers may not only provide the dowloadable files, but also manage information about alternative sources. P2P-clients may not only download, but also provide information for upload at the same time. For access control similar approaches are possible. A site with a complex security policy, may require the client to contact other, different, sites, with each checking a single condition and certifying the approval by sending back a credential.

The statistical program of the department in our example may retrieve additional input from data sources outside the department. However the number of users causes these connections to be a bottleneck on the application server. A performance optimisation may be to require the client to retrieve all external information itself and provide them together with the access request.

Another performance optimisation can be to stop the client PC requesting access, if all required credentials are not present - thus the request would fail anyway.

2.5 Summary

This chapter provided an introduction to the area of distributed system security. It started with an introduction to general computer security to narrow down to specific access control security versus information flow security. Then the overview of network security included a list of specific threats with their countermeasures, which included cryptography. The section on distributed system security discussed the influence of

employing several independent network nodes. The chapter concluded with an example set-up, from which we derived requirements for open distributed systems onto the access control mechanism.

Chapter 3

Access Control Research

This thesis is concerned with the design of a unified access control framework, which is required to satisfy various access control models. This chapter reviews the history of access control research. We will review and compare different access control models, policy specification approaches, as well as access control mechanism implementations.

Early work on protection of computer systems [135, 75, 134] aimed to solve the security problem as a whole, including use of cryptography or denial of service. The previous chapters has given a brief introduction of this topic. Recent research divided the security problem into subcategories, such as secrecy, denial of service attack detection or access control. In this chapter we will review the work in the area of access control.

Table 3.1 lists influential publications in this area and can be used to achieve a basis for advanced research. We will review each of these seminal papers in the relevant category of this chapter. Other authors also aim to provide a comprehensive overview, these include McLean [97], Landwehr [92] or Amoroso [6]. Sandhu sets in "Access Control: The Neglected Frontier" [137] different models into relation.

In the rest of this chapter we will introduce some criteria under which perspectives

1971	Lampson [89]	Access Matrix Model
1973	Bell and LaPadula [12]	Mathematical Foundation BLP model
1976	Harrison, Ruzzo and Ullman [55]	General access control model
1987	Clark and Wilson [27]	Comparison Commercial and Military Security
1989	Brewer and Nash [22]	Chinese Wall Security Policy
1994	Sloman [152]	Policy driven management for distributed systems
1996	Sandhu [142]	Role Based Access Control

Table 3.1: Key Publications in Access Control Research.

the different access control models, prolicy frameworks and mechanism were reviewed (Section 3.1). It is worth noting, that the criteria will be used under two contexts which relate either to the requirements the model or policy has on an access control mechanism, or the kind of abilities a particular mechanism has to fulfil requirements of models in that area. Section 3.2 reviews access control models with respect to the introduced requirements. Following this, Section 3.3 reviews policy-driven access control frameworks, as they are closely related to access control mechanisms. Then Section 3.4 compares various access control mechanisms highlighting how the identified requirements are met. Finally Section 3.5 concludes this chapter with a summary comparison of the different work.

3.1 Criteria for Reviewing Access Control Models and Mechanisms

In this section we present common criteria to evaluate the different access control models and frameworks. We selected the criteria below based on the aim of this thesis to provide a unified access control mechanism. Some criteria, such as the extent of the TCB and dynamics, are derived from inspections of different access control models

and policy languages. Others are based on our prediction of future open distributed systems; the location of the TCB or scaling behaviour are examples.

During the course of our research we noted that decentralization and simplification of administration provided further criteria for analysis. We included the principle of least privilege as a criterion as it was a major driving force for early work on access control models. Although one can argue that other criteria should be included, we think our list provides a good basis to identify the requirements of different access control models in regard to the underlying access control mechanism or the fitness of a certain mechanism to support a wide range of access control models. However there is a drawback in using the same set of criteria for the both purposes, namely the exact meaning of a criteria changes with the setting. For example, in the setting of the access control models, dynamics means whether a model has a dynamic access decision function. It is understood that the more dynamic elements the function has, the harder it is to support by an access control mechanism. Conversely, dynamics in the setting of access control mechanims means how easy it is to support a dynamic access decision function for the relevant mechanism.

We will now introduce each criterion in detail and discuss in the next section how they can be used to compare access mechanisms and to check fitness of a particular mechanism for a particular model.

Extent of TCB: Particularly for access control models it is important to understand which objects are trusted. The access mechanism will need to cater for the protection of these objects. Some models require the storing of labels with each subject and object, while others require large lists of the privileges that different subjects have. From the point of view of access mechanisms, some implementations may not provide a TCB which, for example, can store executable objects, this influences the capability of supporting some access control models.

Locations of TCB: In all cases part of the TCB is on the server. In open distributed

networks, a TCB on the client side does not exist because, by assumption, the client is administered independently and a TCB on top of an *a priori* untrustworthy OS is not possible[1] (see Section 1.5 of [47], or [130]). We will interpret this criterion in two ways: (1) for certain access control models we must analyse, which TCB locations are required; (2) for the access control mechanism we must review the locations on which a TCB can be provided.

Decentralization: Some research in decentralized administration has been done. We understand decentralization as the ability to run the access definition function on different locations simultaneously (in a central scenario one entity needs to provide possibly syncronised input). An example can be an ACL which, if it does not change, can be copied and queried from different places; a model requiring the input history of the user is dynamic and thus requires a central TCB entity to store this information.

Dynamics: Access control models which base their access decision function on the current system state, require the TCB to actively compute, while models using labels solely require static comparison. If the access mechanism is to be implemented in hardware, a simple comparison is easier to implement than a dynamically growing table of prohibited access requests. Existing access control frameworks may also lack the ability to instantiate certain dynamic access control model policies, e.g. AMOEBA does not provide a dynamic access decision kernel (see Section 3.4.1 for a discussion).

Scalability: Today computers are ubiquitous tools in companies and even everyday life. Moreover the advent of personal computers have increased the number of different hosts and users in a network significantly. We will review the models

[1]The current efforts of the Trusted Computing Platform Alliance (TCPA) are in the direction of providing a trusted computing base for some applications in an untrusted environment.

in terms of scalability to compare fitness for large networks, such as the global Internet.

Administration: Some access control research has proposed enhancements on account administration [142]. Delegation may be used to jointly administrate an access control mechanism, and also security policy may allow administration in a team. Graphical administration front-ends may simplify the task by providing a better overview of the full system.

Least Privilege: The principle of least privilege requires that users only gain the minimum access required to perform their duty. We will evaluate if this principle has been followed each of the reviewed access control model or mechanism.

3.1.1 Method of Comparison

The reviewed access control literature consists of various approaches. Access control models such as the Chinese Wall model are included (Section 3.2.3), as well as policy specification languages (Section 3.3.2). One can argue that an access control model is not comparable to a policy language, because a policy language can be used to describe more than one model. For identifying requirements onto the access control mechanism layer a comparison is needed. The criteria presented above allow a tentative comparison, but are not quickly accessible. To facilitate a quick comparison we rated each approach in each criterion on a scale of: how difficult or easy it is to fulfill; extensive needed or not necessary; fit for or not instantiable.

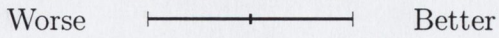


Fig. 3.1: Example Scale of Neutral Value.

Figure 3.1 shows a neutral rating. A rating further to the left expresses for the access control model that the feature is harder to implement, more extensively used or

not provided by the mechanism. The access control mechanism will get a rating far to the right if they provide facilities to instantiate models with a strong requirement in this area.

In the final comparison we chose starplots instead of a flat table, to remind the reader that an exact comparison, such as 'if a certain sanity check is implemented; it is fit to support; or cannot be done'. In general the result is influenced by the fact that not all mechanisms aim to support a large number of models.

The starplots can be generated by plotting the points on the graphs and radiating them around a point, and then linking these points. Starplots are used in CPU benchmarks [53] and a variation can be found in a policy approach comparison by Wies [169].

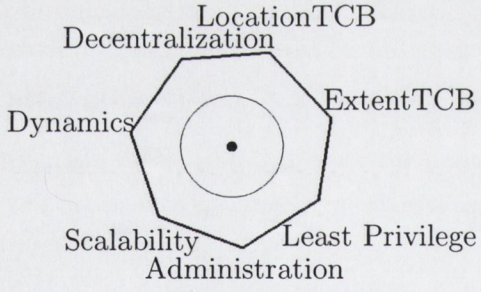


Fig. 3.2: Example of Diagram.

Figure 3.2 shows an example of an ideal access control mechanism. The dot marks the center of the figure, while the small circle depicts a neutral value. A line outside the small value can be seen as positive, while lines close to the center mean harder to implement properties (worst case in the access control model setting). The upper half shows properties directly related to the implementation, while the lower half includes desirable properties, such as value for administration or scaling behaviour.

	Domain 1	Domain 2	Domain 3	File 1	File 2	Process 1
Domain 1	*owner control	*owner control	*call	*owner *read *write		
Domain 2			call	*read	write	wakeup
Domain 3			owner control	read	*owner	

*copy flag set

Fig. 3.3: Portion of an access matrix

3.2 Access Control Models

Early access control models were strictly mathematically formulated. Later on special purpose models were described using graph theory or informal policy description. We will review each category of models by presenting the relevant seminal paper and discuss variations or extensions afterwards.

3.2.1 Access Matrix

Access control in time-sharing systems has been heavily inspired by military security levels [168, 99, 167, 119]. In his seminal paper [89] Lampson presents an abstract model for an access control system. His *object system* consists of an object set X, a set of domains D, and an access matrix A. Lampson explicitly points out that the "domains are objects, and that objects do not live in', or belong to' domains.". Domains can be acted upon by other domains. Figure 3.3 shows an example of an access matrix. Today we call the domains on the left, subjects, while each column belongs to an object. As Lampson's model is centered around the notion of ownership, a domain can own another domain. This abstraction is necessary, as Lampson does not assume a centralized access control function. Each process learns the process, where it is called from and decides using its own discretion if it complies with the request.

Lampson also presents some implementation techniques. He points out that storing the access matrix as a whole is impracticable, but suggests storing each row or column separately. The former is called *capability list* and will be discussed in greater detail later in this chapter in Section 3.4.1. The latter is called *access control lists* and will be discussed in Section 3.4.2.

Extensions and Variations of the Access Matrix

Harrison, Ruzzo and Ullman [55] generalised the access matrix and showed that the safety question is undecidable in non-monooperational systems [2]. This result inspired work on the Take-Grant Model [18, 17], which solved the safety question in constant time using a graph approach. Although this result is beneficial, a drawback is its missing expressiveness. Sandhu developed the TAM (typed access matrix) and ATAM (augmented typed access matrix), which holds the expressiveness of the HRU model, but by introducing types is able to answer the safety question. The ATAM model further allows to test for absence of access rights, as needed in dynamic separation of duty. Ganta's thesis [46] provides an introduction to the group of TAM models.

Assessment of the Access Matrix Model

Extent of the TCB: The access matrix is at the core of the access matrix model. As such, it needs to be protected. An ACL is stored at the object server, while the capability list is stored at the subject. Possession of a capability provides by definition the associated access rights. If capabilities are stored at the client node in open distributed systems, they require additional protection such as cryptographic signatures or encryption.

[2] Systems, which require, in the HRU model, more than one operation to derive the access decision, such as A can read the file if (1) he/she is the owner or (2) the file is readable by non owners.

Locations of the TCB: The paradigm of ownership requires no central TCB, only a TCB at the location of the object server. However, in the original protection paper it is pointed out, "the identification is supplied by the system and therefore cannot be forged." [89].

Decentralization: The access matrix can be stored at the object server and no additional information is required. Therefore, the access decision can be decentralized.

Dynamics: The access matrix model is inherently static. Variations, such as the DTAM model, propose dynamic changes of the access matrix. It requires an active TCB component to incorporate these changes.

Scalability: In large systems the access matrix is of course, very large. In the protection paper [89], it is already suggested that the access matrix is to be stored as access control lists or capability lists. Both of them face scalability problems if the access is fine grained.

Administration: Depending on the definition of the object owner in large distributed systems, administration of a simple access matrix model may be hard. For example, an organization as owner of the objects, which wishes to select on a per user basis, faces an administration problem. If each employee is seen as the owner of their own files, the administration is decentralised, but requires higher level of security awareness of each user.

Least privilege: The access matrix model is able to support the principle of least privilege. However, a fine grained access right partition will result in a large table, which is harder to administer.

3.2.2 Multi Level Security

The first multi-level access control model was developed in the early 1970s at the Mitre Corporation. The Bell-LaPadula (BLP) access control model [14, 6, 12] utilizes work from general systems theory "to formulate a mathematical framework within which to deal with the problems of secure computer systems". The original paper defines the state of a secure computer system as a triple (b,M,f). The informal definitions of each triple value is given below and were taken from [12]

$b \in P(SxO)$: Indicates which subjects have access to which objects in this state

$M \in M$: Indicates the entries of the access matrix in this state

$f \in F$: Indicates the clearance level of all subjects, the classification level of all objects, and the need-to-know association with all subjects, and objects in this state

The formal definition can be found in the [12]. For our discussion we restrict ourselves to a high level view. It is worth noting that a secure system cannot reach an insecure state if it takes input from a set of given requests. The BLP model provides four types of requests [12]:

1. A request by a subject to be granted access to an object in a particular mode;
2. A request by a subject that another subject be given some access attribute with respect to some object;
3. A request by a subject to create an object in the system; and
4. A request by a subject to delete an object from the system.

The first type includes the known read, write, execute or append. The second type spawned the research into delegation. Finally, the third and fourth types of requests

influence the system as a whole by modifying the set of objects. Bell and LaPadula also observed that the interaction of multiple accesses could lead to a security breach. They formulated three principles for secure system design:

Security principle: Also known as the simple security property, it is directly derived from the security definition in military-governmental situations. A higher ranked subject is allowed to read all information at their level or below. Read ups are prohibited according to this principle. Figure 3.4 is taken from Amoroso [6] (page 105) and reproduced to show the simple security property in non-mathematical terms.

Interactivity principle: Also known as $\star$-property prevents a subject to write to objects of lower security level. As a result the confidentiality of information can not be broken by a high level subject through writing it to low level objects. Figure 3.5 taken from Amoroso [6] (page 105) demonstrates the $\star$-property.

Tranquility principle: Requires that classifications of active objects do not change during normal operation. This prevents a user reading at a different clearance level as they write.

Least Privilege: The principle of least privilege can be followed in the access matrix model.

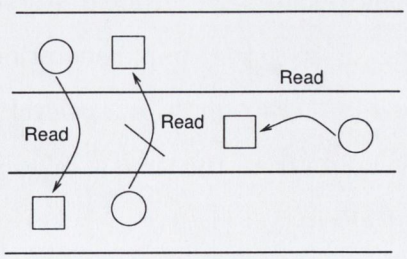


Fig. 3.4: The Simple Security Property of the BLP Model.

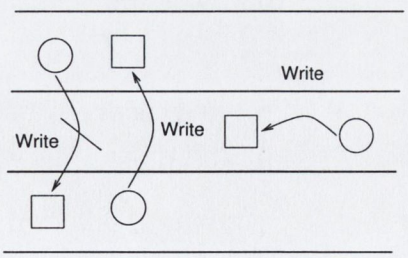


Fig. 3.5: The $\star$-Property of the BLP Model.

Extensions and Variations of Multi Level Security

It has to be noted that the BLP model itself is a group of several models refined over time. In a later version [13] a discretionary security property was added as an axiom to the original purely mandatory model. In fact, the BLP model was so influential that other security models were cast into the BLP model, such as the Chinese Wall model (described in Section 3.2.3) [136] or secure information flow [33].

Biba at the MITRE corporation identified confidentiality as the main concern of the BLP model. He designed a group of models, where the main concern was integrity [16]. Reading information at lower levels is understood to dilute the integrity of high-level subjects. This results in a model of the same style as BLP, except that the rules are roughly opposite to those of the BLP. Two variations of the Biba model are interesting in that they introduce dynamics into the access control model. The Subject-' and the Object-Low-Water mark model do not prohibit a read down (or write up), but consecutively lower the subjects' (objects') integrity levels. Because in both models no measures allow the reverse process, the integrity level changes are monotone and ultimately all subjects (objects) will share the same, lowest integrity level.

Assessment of the Multilevel Security Models

Extent of the TCB: The security labels need to be protected. Further on the 'confinement problem' [90] caused academic debate. In an ideal system, low-level processes cannot read high level data. In an actual implementation, by use of covered channels, this may still be possible. For example, a low-level process may deduce information about the high-level data from the time it takes until a reply is given (also known as side channel attacks).

Locations of the TCB: An ideal mandatory access control system requires that high-level subjects not be able to write to low-level objects and that accord-

ing to the tranquility principle a subject not be able to change its security level between read and write accesses; consequently, mandatory access control requires a TCB on the client side to control these requirements.

Decentralization: A simple BLP model does not change the security labels, which allows the storage of the labels to be decentralized. Although some models, like the Object-Low-Water mark, require security labels to be changed dynamically, these labels have to be stored centrally.

Dynamics: The low-water mark and similar models require the labels to be changed throughout consecutive accesses. This requires an active entity rewriting the affected labels.

Scalability: Each subject and object has a single label and access rights can be computed from these labels; thus the BLP model does not face scalability problems.

Administration: Administration of the BLP model is simple once the semantic of each security level has been established.

Least Privilege: All mandatory access control models face problems with the principle of least privilege. A top secret ranked officer working on one project still gains access to secret data of other projects. A solution by typing has been proposed by Sandhu [141].

3.2.3 The Chinese Wall Model

Inspired by the division between military and commercial security, as pointed out by Clark Wilson [27], Brewer and Nash proposed an access control model useful for market analysts advising financial institutions. The Chinese Wall model is best described

informally. At the beginning a user (e.g., analyst) is not restricted in accessing any object. Once the first object in a conflict-of-interest domain is accessed, an imaginary wall is built - preventing access to any other object than the first one. During the course of actions, the user's access is restricted further and further, resulting in the final state where each user can only access exactly one object in each conflict of interest domain. The reasoning behind this policy becomes clear if the objects are thought of as insider information of, say an oil company, then the conflict of interest domain includes all oil companies. Thus, market analysts are allowed to advise one company, but only if they do not have insider knowledge on other companies in the same field.

Chinese Wall Model Related Work

In the original paper, Brewer and Nash argued that their model "cannot be correctly represented by a Bell-LaPadula model." [22]. Later Sandhu [136] demonstrated, by honing down the definition of subject, how to represent the Chinese-Wall model as a lattice. A subject, such as a user, will be represented in their definition by different principles, while each principle is statically linked to a conflict of interest domain. The user's labels represent which principles they can activate. The special label of top element is called SYSHIGH and cannot be assigned to a subject, because such a subject would break the Chinese-Wall model. If a user wishes to access a new conflict of interest domain, which would result in the activation of a new principal, the label update takes place and, if the user needs to get the label SYSHIGH assigned to activate the new principle, the access will be rejected.

Assessment of the Chinese Wall Model

Extent of the TCB: The full access history of each user needs to be stored.

Locations of the TCB: No TCB is needed on the client, but the information about past accesses have to be stored inside a TCB.

Decentralization: Decentralized computing of access is not possible because the access decision is based on past input. However, each conflict of interest domain and each client can use a different central server.

Dynamics: Before an access request, the 'walls' need to be dynamically updated, which requires an active computation.

Scalability: There are two potential scaling factors in the Chinese Wall model: the number of clients, and the number of companies. Both can cause scaling problems, but it is expected that the number of companies will not grow rapidly.

Administration: The Chinese Wall model does not require an external administration as the walls build themselves up while users interact with the system.

Least Privilege: The notion of *restriction to eradicate the misuse of insider knowledge* opposes the principle of least privilege. A user in the Chinese Wall can access all information, except if previous accesses require restriction.

3.2.4 Role Based Access Control

Role based access control (RBAC) is the fourth major model in access control, although it is claimed that RBAC is policy neutral [142]. The root of RBAC traces back to Clark and Wilson's work [27] of requirements on commercial security systems. A study of 28

organisations conducted by the NIST [42] found that the access control decisions are based "on the role that individual users take on as part of the organizations." Other work added the *principle of least privilege* [135] and *separation of duty* [27, 112] to this. Analysing the user grouping mechanism found in UNIX and other operating systems, and privilege grouping found in some databases [10, 32], different researchers proposed models for RBAC [41, 45, 71, 81]. Two groups were particularly active Nyanchama & Osborne [122, 123, 121] and Sandhu [138, 142, 140, 139]. We will shortly summarise Nyanchama & Osborne's work and describe Sandhu's RBAC96 model family in greater detail. This seems reasonable because Sandhu was part of the team working on the NIST standard for RBAC [42, 114].

Nyancham & Osborne developed a formal model for role based access control [122]. After pointing out the differences of a privilege to a capability, they give definitions for each element, such as privilege, roles and role relationships. They identify the need for common privilege and privilege augmentation - roles that combine the privilege of two or more junior roles. At the heart of their RBAC model is a role graph model and role graph maintenance algorithms. The role graph model for role organization can be converted into a tree (hierarchy) and vice versa. The role graph maintenance algorithms include addition, deletion and split (partitioning). Their work presented a firm formal base for role based access control and complemented Sandhu's work about properties of RBAC models [142] in relation to practical experience data of the NIST [113].

Sandhu presented the RBAC96 model family shown in Fig 3.6 (taken from [142]), which consists of four models. $RBAC_0$ is the base model providing the concepts of 'user to role' (UA) and 'privilege to role' assignments (PA). Figure 3.6 all parts which are not marked otherwise, belong to $RBAC_0$. These are:

U, user: A user generally is a human being, but can also be an independent acting agent. A user can own many roles.

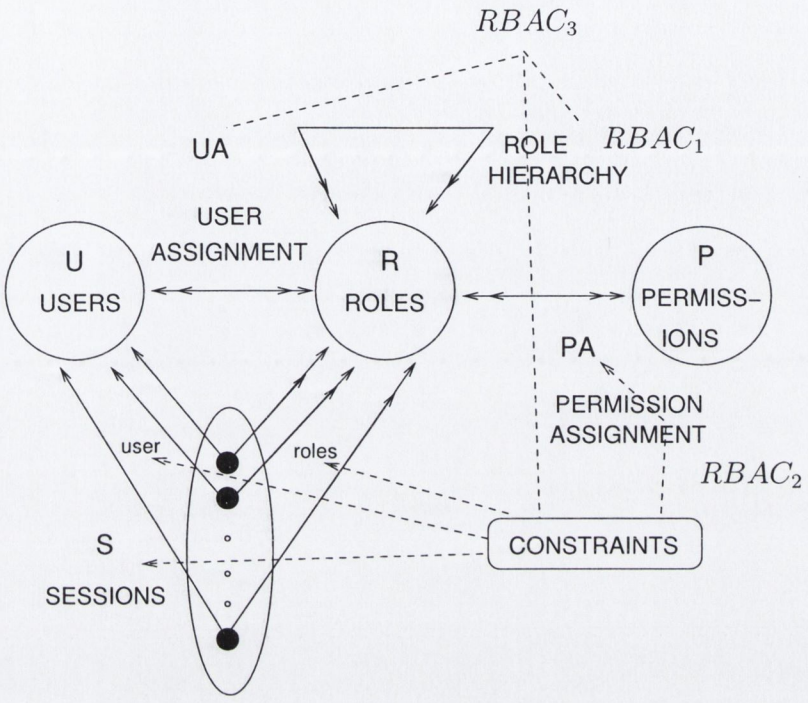


Fig. 3.6: Family of RBAC Models.

R, roles: A user performs a task in practising a role. A role combines all permissions necessary to perform its duty.

P, permissions: Permissions are the access rights to perform a certain action or request. $RBAC_0$ does not use negative permissions, which are modelled as constraints in $RBAC_2$.

S, sessions: During a session, a user activates a role or a subset of roles. Each session is associated with a single user and the set of roles associated with it is constant.

$RBAC_1$ adds role hierarchies (RH) to $RBAC_0$. A role can now own another role with the permissions inherited transitively, e.g. the role of a head of a financial department includes roles of signing accountant, counter signing accountant and employee of the company.

$RBAC_2$ adds constraints to $RBAC_0$. A constraint is a negative permission, restricting the use of a role or permission. Constraints can apply to each element of the $RBAC_0$ model. Examples of constraints can be a maximal spending volume for a signing accountant or mutual exclusiveness of two roles.

$RBAC_3$ combines role hierarchies with constraints. Figure 3.7 from [142] shows the relation of the different models to each other.

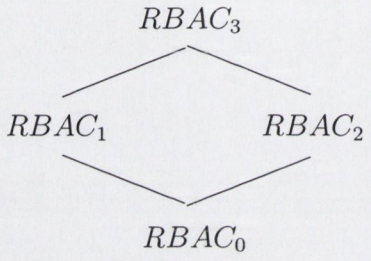


Fig. 3.7: Relationships among RBAC Models.

Assessment of Role Based Access Control

Extent of the TCB: The RBAC model has two mapping functions (user to role and role to privilege), which need to be protected. Furthermore, during operation user-role-session bindings have to be protected.

Locations of the TCB: The RBAC does not require a TCB on the client side, but constraints may need a trustworthy monitoring entity.

Decentralization: It is possible to use RBAC as an administration interface and build an underlying static access matrix. The storage of this access matrix can be decentralized.

Dynamics: The session management at RBAC requires active computation from the TCB, but the mapping, including role hierarchies, is intrinsically static.

Scalability: RBAC has been designed with scalability in mind. The overview can be kept easily as high profile roles inherit rights from junior roles, computation-wise inheritance can be implemented using pointers and iterative privilege evaluation.

Administration: By the use of RBAC administration is greatly simplified, because the semantic of roles allows a clear understanding of privileges associated with it. RBAC allows the decentralization of administration of user to role and role to privilege mapping.

Least privilege: By creating an accurate role for each organizational duty, the principle of least privilege is well enforced.

3.3 Access Control Policies

The access control model describe properties of the full access control framework, for example the integrity of the objects has to be kept. Typically an implementation consists of a static access decision function, which is intertwined with the access control mechanism, e.g. ACL or Unix. The consequence is that a system becomes complex and hard to verify. Research in access control policies solve this problem by providing a unified way to specify different access control models and even general systems management [152]. The system administrator is able to specify and maintain policies for all aspects of information system management. In cases where the organisation's security requirements change, only the policy specification has to be rewritten [169]. Policies are specified in an easy to understand form, but the formal character of the policy language has additional benefits:

- Identification of inconsistencies;

- Conflict resolution;
- Company business policies can be semi-automatically translated into security policies.

For the last point, policy research has proposed different abstraction layers or hierarchies [96, 106, 169], which assist in refining high level company policies down to low level, technical security policies.

For the purpose of this thesis, policies are interesting from another perspective, namely which demands the policy framework sets on the access control mechanism. It has been noted that not all access mechanisms are capable of supporting the full expressive power of policy frameworks [60]. We will review in this section some policy research with respect to requirements onto an underlying policy enforcement mechanism. For this reason the work has been divided into three categories:

- Logic-influenced policy research
- Policy specification languages
- Research regarding the properties of policy systems

3.3.1 Logic-Influenced Policy Research

The first attempt aimed at providing a general framework for expressing authorisations was made by Woo and Lam [171]. Their framework used default logic, which is very expressive. However, it was claimed that default logic is not semi-decidable (neither undecidable) [69].

Ong and Lee [127] proposed a logic model to maintain consistency of bureaucratic policies. Figure 3.8 shows their definition. A bureaucratic fact (BF) can be '*teaching assistant*('*John*');'. A bureaucratic rule (BR) has the general form *conclusion* ←

A logic model for bureaucracy is a triple:

$$\langle \mathbf{BF}, \mathbf{BR}, \mathbf{BIC} \rangle$$

where

1. **BF** is a set of ground formulae representing bureaucratic facts.
2. **BR** is a set of deductive rules of the form:

   $$H \leftarrow B_1 \& \cdots \& B_n \text{ where } n >= 0$$

   where $H, B_1, \cdots, B_n$ are atomic formulae and H does not have any predicate symbols in **BF**.
3. **BIC** is a set of integrity constraints of the form:

   $$false \leftarrow B1 \& \cdots \& Bn \ where \ n >= 0$$

   where $B_1, \cdots, B_n$ are atomic formulae.
4. The logic model *is* **consistent** iff all integrity constraints are satisfied:

   $$BF \cup BR \vdash c \text{ for } \forall c \in BIC$$

   where $A \cup B$ means we can prove B from A.

Fig. 3.8: Definition of the Ong-Lee Model.

conditions, where *obligation, permitted, forbidden,* and *waived* are the possible conclusions. Thus a BR example can be

$$obligated(university;\ insured(X)) \leftarrow employees(X);.$$

The logic model can be used to detect inconsistencies using an abductive unification algorithm.

A Model Based on Deontic Logic

A model based on deontic logic has been proposed by Jonscher [76]. Deontic logic is a modal logic with predicates for obligation and prohibition. Jonscher notes that besides classical access rights, deontic rights in access control are required. He introduces duties and liberties (both in positive and negative form). Duties require the user to do something, while liberties give the user the possibility to do something. Jonscher points out that there are cases in which a simple access right prohibits the user from an action, which a positive duty requires him to do. Part of the contribution is a framework to solve these conflicts. Each policy, or (BR), has three main parts (Event, Condition, Action). Events are divided into:

- Data manipulation events, such as begin or end of operations on data
- Time events, either absolute or relative time
- External events, raised explicitly by applications
- Transaction events, such as begin or end of transaction, abort, commit

Additional to these simple events, complex events are introduced - combinations of events. It is worth noting that the notion of context is reflected in their model as situations, which are defined as a combination of an event and a condition. Again, complex situations are defined by combination of situations

$(s_1|s_2)$: Two situations may independently trigger the complex situation.

$(s_1; s_2[> t_{max}])$: A sequence of events (maximal t_{max} apart).

$(s_1, s_2[> t_{max}])$: A sequence of events, with arbitrary order of occurrence (maximal t_{max} apart).

The notion of roles and tasks allows the behaviour of users in organizations to be modelled. For further details, the interested reader may consult the original text [76]. Here it is important to note the different events, and that the relation of events needs to be captured by an underlying access control mechanism.

Authorisation Specification Language

The work of Jajodia et al [69, 70] is motivated by separating policy enforcement from policy decision. An Authorisation Specification Language (ASL) is proposed. The authorisation policy maps a 4-tuple (o,u,R,a), where 'o' is an object, 'u' user, 'R' a role set, and 'a' an action, to the set {authorised, denied}. ASL is a logical language created from an alphabet of constant symbols, variable symbols and predicate symbols. Predicate symbols (taken from [69]) are:

cando: Authorisation explicitly inserted by the system security officer.

dercando: Authorisation derived by the system using logical rules of inference.

do: An authorisation that holds for each subject on each object.

done: Done rules represent the accesses executed by requestor subjects.

active: Is used to capture the role of active role/s for a user.

dirin and in: They capture the direct and indirect membership relationship between subjects.

typeof: Captures the grouping relationship between objects.

error (integrity rule): If error() can be derived through some rule, then there is an error in the specification or use of authorisations due to the satisfaction of the conditions stated in the body of the rule.

Using these predicates, different conflict resolution strategies are discussed and examples of authorisation models are given. It is worth noting that the integrity rule allows the expression of dynamic constraints, such as those of the Chinese Wall model.

$$\begin{aligned} error() \leftarrow\ & done(o', u, R, a', t) \& done(o, u, R', a, t') \\ & \& typeof(o, Company - A) \\ & \& typeof(o', Company - B). \end{aligned}$$

Throw an error, if user u has accessed an object of type A and another of type B, where A and B are in the same conflict of interest class. Thus a policy enforcement mechanism for ASL has to be able to generate 'events' not only when an access request is given, but also after finished.

Assessment of Specification Languages Influenced by Logic

Extent of the TCB: The access policies need to be stored in a protected area, as well as all preconditions and access system facts (e.g. BF of Ong Lee's model).

Locations of the TCB: There is no apparent need for a TCB on the client side, but the evaluation of the logic formulae has to be done at a TCB. Further TCB areas beside the server may be required if, for example, external events need to be generated.

Decentralization: In case of Ong Lee's model [127] it is possible to decentralise the access decision function, if all facts and policies are known in advance. However, in a system like Jajodia's [69], which includes variable symbols, a central server needs to maintain these. Some event sources of Jonscher [76] may also require a central entity.

Dynamics: All policy systems influenced by logic need to infer additional formulae, until the final access decision is reached - this always requires an active TCB.

Scalability: Logic-influenced access control systems can have scaling problems if the number of policies makes computations overly expensive. This was solved by introducing grouping.

Administration: One motivation for using logic in access control is to be able to automate reasoning about the final system. This can point out conflicts or inconsistencies in the specified policies.

Least privilege: Access policies have the potential to model the principle of least privilege accurately. One strategy may give each user a rough set of rights, thereafter large areas of rights will be prohibited. The system will point out conflicts, which can be resolved using the principle of least privilege.

3.3.2 Policy Specification Languages

The aim of policy specification languages is to allow specification of policies for system management in a uniform language. One advantage is that the company's policy

can be described in one formalism and employed in different environments [25]. Additional benefits can be gained from the use of policy editors [95], which make policy specification simpler and more intuitive for the naive user.

The idea of taking advantage of graphical policy representation has been shown at the Miró set of visual languages [59]. The authors describe an instance and a constraint language. In order to answer the questions "Which users have which kinds of access to which files?" and "Which of all possible user-file accesses are realizable by the operating system and acceptable according to our site's security policy" [59].

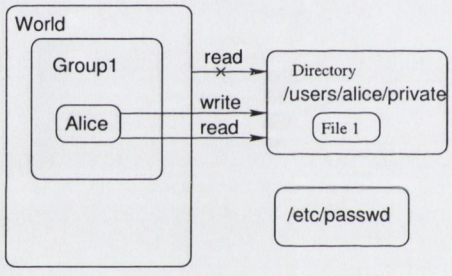


Fig. 3.9: A Simple Instance Picture.

The instance language uses boxes and arrows to depict access rights. A box, which does not contain other boxes, depicts a user or file. Boxes can be contained in each other to depict hierarchies, such as files in a directory or users in a group. A labelled arrow depicts the granting of access rights. A simple instance picture is shown in Figure 3.9. User Alice has *read* and *write* rights to the files in her private directory. The world, that is all users, explicitly do not have read access, depicted by the crossed out arrow. The same policy could be described by leaving out any arrow from world to '/users/alice/privat', but this may not resolve conflicts in more complex scenarios. The authors propose the notion of a witness set, which can resolve conflicts. The syntax and semantic of a witness set, as well as the other elements of the Miró language are fully formalised in [59, 161].

The constraint language allows the drawing of pictures (patterns), which depict constraints caused by either the operating system or higher-level policies. Constraint pictures need to be respected by the instance picture (policy). Informally speaking, a pattern in the constraint picture needs to embed into the instance picture. The work of the authors includes a policy editor, ambiguity checker, constraint checker and back-end tools to enforce the policies. However it was pointed out, that it is NP-complete to determine whether an instance picture satisfies a particular constraint or not.

SPL[3] is a security policy language with complex constraints. Ribeiro et al. [25] present the structure, as well as basic language constructs and an implementation. SPL consists of the following constructs:

Entities: SPL entities are typed objects with an explicit interface. Entities can be internal or external. External entities are understood to be entities outside the TCB. It was pointed out that external entities are necessary in some (very flexible) access control models.

Sets: Entities can be collected into sets. There are two types of sets: categories and groups. A category joins entities with a similar property. Groups are formed by explicit insertion and deletion.

Constraint rules: The core unit of SPL is its rules. These form constraints onto system execution. Rules are comprised of two logical binary expressions, one to establish the domain and another to decide on the acceptability of the event. A special tri-value algebra derives from the two binary expressions the result of the rule. Hereby the three values are *allow, deny* and *not apply.*

Policies: Policies are the group of rules and sets. Policies can be closed or open. Closed policies deny everything, which is not explicitly allowed, while open policies allow all, except certain denied actions.

[3]security policy language

SPL knows some special constraints: historical and obligation. Historical constraints require the existence of certain events in the past event list. Obligations allow a current event, but require another event to take place in the future. Because this is not enforceable (i.e, the system may stop before the future event happens) a closer definition of the use of obligations needs to be given [30]. An obligation can only be inside an atomic request. Atomic requests are requests which include several simple actions, each of them will be run before the request finishes. If not all the actions of an atomic request can be fulfilled, a rollback takes place. A certain type of actions,"real actions" (e.g. printing a file) cannot appear inside atomic requests, because they cannot roll back. The implementation of SPL handles both historical and obligation events in a similar fashion, while relying on the application to order simple events in an atomic request to comply with the obligations at time of event request. Historical based events require a monitor-like security service recording relevant events and providing proof of compliance to requesting services.

One of the most refined policy language has been developed at Imperial College London. The Ponder policy specification language is motivated by different research efforts into (I) policy hierarchies [106], (II) specification of obligations [102], and (III) system management [152]. Ponder has been used in different areas, such as firewalls, operating systems, databases or Java. The expressive power of Ponder is achieved by several grouping and constraint constructs. These also facilitate object reuse.

Ponder provides positive and negative authorisation policies. A new policy can be instantiated directly or first be declared as a new policy type, and subsequently this type can be instantiated in different domains. Information filtering also facilitates object reuse by transforming input or output parameters of an action to suite another actions format. Delegation allows temporary delegate access rights and is realised by allowing a grantor to delegate (temporary insert authorization policies) for the grantee. Similar to negative authorization policies are refrain policies, which are meant

to be enforced by the subject rather then the target system. A deployment model for Ponder [36] explains the concept of enforcement agents residing on each node (e.g. target and subject system). Refrain policies therefore seem to require a TCB on the client side, but it becomes clear that refrain polices express the self-motivated desire of the subject to refrain from doing certain actions. Hence the TCB on the client side does not belong to the system wide TCB, as in mandatory access control models. Obligation policies in Ponder specify obligations of a manager, which is part of the TCB, at occurence of a certain event. Thus, an event monitor is required.

Constraints can be part of every policy in Ponder, additionally composite policies can have meta-policies as a special constraint. Meta-policies may specify that two policies cannot be applied at the same time. Composite policies are either policy groups or roles. Groups simply join different policies into one group. Roles are based on the semantic meaning of combining different policies required to perform a certain duty. Roles can be defined via role types, which then can inherit policies from superior roles.

Assessment of Policy Specification Languages

Extent of the TCB: The policy specification needs to be protected. In case of historical dependent policies, an audit lock needs to be kept.

Locations of the TCB: Policy languages do not specify whether mandatory or discretionary access control models will be instantiated. Policies like the refrain policy of Ponder may require a kind of enforcement agent on the client side. This enforcement agent would be a TCB alike protected entity, which restricts the requests that local applications may send out.

Event-based systems require further TCB entities for the event infrastructure.

Decentralization: Both incorporation of events or access request history will require a central trusted entity, prohibiting full decentralized enforcement.

Dynamics: The scaling advantages of policy-based designs are achieved through reuse, grouping and policy hierarchies - all these require active TCB parts, which resolve the dependencies.

Scalability: A design goal of policy languages is scalability, which is achieved by grouping and reuse of policies. One scaling issue can be the number of different policies, especially considering the evaluation process if the policy is applicable, this can be solved by restriction and constraints. Another scaling factor will be event logs, which can become large.

Administration: Policy languages facilitate administration by providing a unified interface - the policy language. Policy editors can aid the administrator by visually showing dependencies. Conflict resolution can be done semi-automatically, which is helpful in large complex systems.

Least privilege: Policy languages do not directly support the principle of least privilege, but rely on semantics of roles and a correct systems policy specification.

3.3.3 Research about Properties of Policy Systems

This section discusses different influential research papers about properties of policies. The subject of each paper is fundamentally different, so that we will split the discussion into different subsections.

Obligations

Minsky and Lockmans paper on "Ensuring Integrity by Adding Obligations to Privileges" [102] was an early and influential paper on obligations. They pointed out the need of a system to transit through *improper* states to achieve the next proper state. In databases, one solution to handle these transitions was to hide them in an abstraction, called *transaction* [50]. Transactions are convenient if the number of transitions is limited. In systems which require a greater flexibility, a transaction would not offer the same freedom as the mechanism of obligation, which Minsky and Lockman proposed.

Obligations do not directly prohibit an action, but impose a certain duty on the system (user). This duty is required to make sure the system will eventually reach a proper state again. Where an access request may not always result in an improper state, obligations can be triggered by events. An obligation will have a requirement, deadline and a sanction. If the requirement is not fulfilled by the deadline, the system will impose the sanction. Sanction can be one of the following:

- Correction: The system will be corrected to reach a proper state.
- Roll-back: The system rolls-back to the previous proper state.
- Emergency Measurement: If no proper state can be reached normally, an emergency state may be employed to bring the system back.
- Imposition of another obligation: This can be seen as a penalty if the first obligation was ignored. The system may now no longer act as generously upon that actor.

The authors identified that some obligations may be implemented as post-conditions, while others require the creation of a *critical-mode*, which is entered if an obligation is imposed. The critical mode may only be left if all obligations are fulfilled, preventing

the system from being overthrown by the interaction of multiple access-violations.

Assessment of Requirements of Obligations

Extent of the TCB: In the case of obligations, a protection of the state of the obligation needs to be in place.

Locations of the TCB: It is rare that obligations will be present only at the server or client side, hence the TCB, which enforces obligations, will reside at different points in the system. The TCB may include an event monitor and system state storage to enable a roll-back mechanism.

Decentralization: Obligations by their nature affect the full system, hence their enforcment cannot be decentralized.

Dynamics: An obligation always requires the comparison of a current state with a requirement. In most cases a requirement will depend on other system variables rather than a simple value. Hence obligations require an active TCB.

Scalability: Obligations may inhibit scalability, because large systems may not be able to roll back. Or the number of simultaneous violations, which are enforceable by obligations limit the growth.

Administration: Obligations can simplify administration, because not every transition needs to be foreseen. Obligations can be seen as allowing the system to adapt to situations by temporary misbehaviour.

Least privilege: Temporary allowing an access violation does not keep to the principle of least privilege.

Policies as System Objects

An exploratory discussion of the representation of policies as system objects is given in [105]. Understanding 'Policy as objects' allows the manipulation of them like other system objects. This also allows managers and policies from different domains to interact. For example, one domain may have a policy object, which has the power to derive certain information, while another domain has a policy object which may require that information. If the policies (as objects) are able to act and be acted on, it is possible for the motivated policy object to interact with the authorised one to achieve its task. Moffet and Sloman [105] introduced a modality for policy objects. A policy may have the following modalities:

- Permitting (positive authorisation)
- Forbidding (negative authorisation)
- Requiring (positive motivation)
- Deterring (negative motivation)

This allows policy objects to be split into two groups: authorisation policies and motivation policies. An investigation shows that the most common authorisation policy is access control. Motivation policies describe high-level goals, such as "The manager of department D is to ensure that the department can always recover from media." Interaction of motivation and authorisation goals thus allows the administration of low-level systems via high level goals. Moffet and Sloman identify some interactions between policies (policies as collections of policy statements, hierarchical policies, ordering of policies), which were partly influenced by Holden [63]. However, the authors noted that a precise definition of these interactions were one of the outstanding issues.

Assessment of Policies as System Objects

Extent of the TCB: The policy object as such needs to be protected from unauthorised manipulation.

Locations of the TCB: The two types of policies (authorisation and motivation) will most likely reside in different parts of the distributed system. One server will handle an authorisation policy while motivation policies belong to the management domain and may be stored on an administration server.

Decentralization: Because policy interaction requires that policies from one domain access those in another, it is unlikely that all policies can be stored at all local places, which stops decentralization.

Dynamics: The concept of policy objects, which can be acted upon, is inherently dynamic.

Scalability: Policy objects facilitate scaling by reducing the number of policy evaluations through introduction of policy interactions, such as collections of statements or hierarchical policies.

Administration: Administration is simplified because high-level goals can be implemented as motivational policies, which interact with low-level authorisation policies.

Least privilege: The concept of policies as objects is neutral to the principle of least privilege.

3.4 Access Control Mechanisms

This section reviews related access-control mechanisms and full frameworks. We will use the same criteria that we used in the previous section, but we shall interpret them from the point view of whether the relevant mechanism is able to support access-control models, with a strong demand in that requirement.

Firstly the group of capability-based designs will be reviewed, followed by a group of inherently different approaches. Finally, certificate-based frameworks will be discussed, together with the influential Kerberos authentication framework. The latter is relevant also to the requirements for emulating Kerberos services.

3.4.1 Capability Based Designs

In his access-matrix model paper [89], Lampson noted that there are two ways for implementing the access matrix. One of them is capability lists (or c-lists), the other ACL lists, which is reviewed later in this chapter. In a seminal paper from 1974 [39], Fabry presented "capability-based addressing" as a method of addressing shared-memory segments. His design would implement capability-based protection on the hardware layer, which was later iterated in different systems [116, 62, 64], up to the Intels Iapx 432 processor [110]. Karger and Herbert proposed "an augmented capability architecture to support lattice security and traceability of access" [79]. This was disputed by Boebert in "On the inability of an unmodified capability machine to enforce the *-property" [21]. Boebert showed that, since capabilities bear the right to gain access by simply possessing them, it is possible to delegate access rights in a discretionary fashion, even to unauthorised subjects. Consecutively, Li Gong introduced the identity-based capability system (ICAP) [48], which prohibited delegation by binding capabilities to user identities. A simple copy operation no longer was sufficient.

Capability research continued to explore different designs based on operating sys-

tems. The research led to valuable insights for early research networks, where each computer was owned by the same administrative power. Today's networks consist of a collection of independent nodes. Operating systems using capabilities as protection mechanisms include EROS [151], which pointed out the need for flexibility on the security-model layer. There are many systems based on OS approaches or that assume a TCB on all nodes [156, 2, 31]. Some aim for true distributed computing by providing location transparency or shared memory among the nodes [164, 101, 43, 165, 104]. Kain and Landwehr's taxonomy for capability-based systems [78] is worthy to note. It attempts to classify all systems by introducing five multiple-choice questions. Finally, Levy [93] presents a comprehensive survey of early capability systems.

The AMOEBA Distributed Operating System

We will review the AMOEBA and ICAP systems because they are representative of the different capability-based approaches. In the AMOEBA distributed operating system [157, 108, 107], Tannenbaum and Mullender extended the capability idea with networked operation. They assumed a fall in hardware costs, and consequently proposed a multi-node distributed OS connected through a network. The intruder was assumed to have full control of a node, but not of parts of the client or server computer. Capabilities were used to protect ports to message channels, which lay at the core of IPC[4]. A server has one or more ports, and knowledge of a port grants the right of communication with it. Access to the network is governed by F-Boxes, which are effectively TCBs, in order to prohibit an intruder from impersonating a server. An impersonating of a server occurred in the man-in-the-middle-attack, first shown by Lowe [94] for the Needham Schroeder authentication protocol. To listen to a certain port, the server tells the F-Box a secret number G. The F-Box computes a one-way hash function $P = F(G)$ and listens on port P. The server distributes P by means of

[4]Inter Process Communication

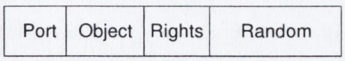


Fig. 3.10: A Capability in AMOEBA.

an access-control policy. A client knowing P can send the server messages by doing *PUT(P)* on his F-Box. The F-Box will not transform P further, while the receiving F-Box delivers the message to the server. An intruder cannot receive messages by doing *GET(P)*, because its F-Box would listen on port $P' = F(P)$.

A full capability of AMOEBA is shown in Figure 3.10, port is the port (or P) the server listens on, object is the object ID, rights the requested access rights, and random is $RANDOM = F(random\ number\ XOR\ rights\ bits)$.

Identity Based Capability System

The identity-based capability(ICAP) system [48] uses capabilities as the access mechanism, but protects propagation of capabilities by ACL lists. Li Gong uses access-control servers (ACS) as administrative servers, which are separate from the object servers. On the ACS, the system stores the internal capabilities. An internal capability consists of the object identifier and a random number. The external capabilities include the object identifier, the access rights and also a random number. The two random numbers are not the same, but the one in the external capability is derived from the one in the internal.

$$Random1 = f(C1, Object, Rights, Random0)$$

$Random1$ denotes the random number in the external capability, $Random0$ the one in the internal. The function f is a trapdoor function. The function takes the rights, object identity and ID of the capability owner into account. Therefore, if a capability was shared by the rightful owner, the act of sharing will be detected. The ICAP system provides delegation. It does this by allowing the owner of a capability to sign

a delegation request and pass it, together with the capability, to the new user. At the first access request–provided the mandatory access-control policy does not prohibit the delegation–the new user may exchange the delegation request for an own external capability. The benefit of the ICAP system is that access requests are based purely on capabilities, an arrangement that enhances performance, while delegation and mandatory access-control are supported using ACL on the access control server.

Assessment of Capability Based Access Mechanism

Extent of the TCB: AMOEBA provides direct protection for the capability, which depicts the right to access. The ICAP system protects the capability using a hash function. In most capability-based designs, capabilities are protected by a TCB on each node. Thus a user may not randomly construct a capability from scratch, lest brute force break the security of the system.

Locations of the TCB: Pure capability-based designs require capabilities to be protected. Therefore, a TCB on each node, including the client PC is required. This may simplify instantiation of certain access-control models, but it breaks a core assumption of the work described in this thesis.

Decentralization: Capability-based designs facilitate a decentralised access-control function, because a server has all the information necessary for to deciding whether a presented capability grants access to its resources. However, the propagation of access can be a problem as mentioned earlier.

Dynamics: AMOEBA does not provide a reference monitor or active entity for calculating dynamic access-decision functions. The ICAP system provides this entity by the access-control server. In case of obligations, the access-control server can

send out events or update the delegation policies dynamically. In both cases, the revocation algorithm may restrict timeliness.

Scalability: Through their decentralisation, capability-based designs provide, optimal scaling properties. For example, there can be several entities computing new capabilities, the number of users is unlimited, and the number of access rights is limited only by the length of the random number of the object.

Administration: Pure capability-based designs do not include an administration layer. Instead they leave the policy by which capabilities are distributed open to each application or system.

Least privilege: By binding a single access right to one capability, and by being inherently a closed-access system [5], capability-based systems allow one to follow the principle of least privilege.

3.4.2 ACL Based Designs

The other approach to implementing Lampson's access matrix [89] was by access-control lists. Access control lists are stored at the object and list each of the subjects with their access rights. Two major operating systems, UNIX and Microsoft Windows, use ACLs in their cores. UNIX divides the subject into three groups (owner, group, other). Each group can get different access rights to the object. There are three (four[6]) rights in the UNIX ACL implmentation: read, write and execute. The most commonly employed access-control mechanism goes back to the first Unix implementation and the security research comunity has proposed various enhancements [19, 44, 128]. Microsoft

[5] all access that is not explicitly allowed is forbidden

[6] The ACL allows a suid bit set, which lets programs effectively execute under a predetermined user-ID rather than that of the caller.

Windows NT[7] was developed later and its access control mechanism can be seen as more modern. Let us now review its implementation.

Windows NT Access Control Mechanism

The central database for the Windows NT configuration data is the *registry*. Besides user preferences the registry holds license keys, passwords, and user-IDs to group mappings. Windows differentiates between five groups: administrators, system, users, creator/owner, and everyone. Gollmann [47] lists the following permissions:

- Read Only (user is allowed to read the key but cannot make changes)
- Full Control (user may edit, create, delete, or take away ownership of the key)
- Special Access (users can be granted permissions according to a specified list)

A typical list for special access would include:

- Query Value: read the value of a key
- Set Value: set the value of a key
- Create Subkey: create a new subkey within an existing key
- Enumerate Keys: identify all subkeys within a key
- Notify: receive audit notification generated by the key
- Create Link: create a symbolic link to a key
- Delete: delete a key
- Write DAC: modify the access-control list for the key
- Write Owner: take ownership

[7]Microsoft Windows NT is a trademark from Microsoft

- Read Control: read security information from within the key

The keys are grouped and sub-grouped, namely into areas of application, and access is granted accordingly. The file system, including the network file system, is based on a similar scheme allowing to grant predefined access rights to user defined groups.

Assessment of the Windows ACL Based Access Mechanism

Extent of the TCB: The registry at the heart of the access-control mechanism provides protection for application and security data.

Locations of the TCB: Each network node has its own registry, which is assumed to be part of the TCB. At the file system security, a domain controller provides central authority to govern access across network nodes in one security domain and group membership translation across security domains. However, the TCB of each client node is trusted (to report the correct user-ID for cross-node file access).

Decentralization: Windows is built on a domain structure, which employs a domain controller as a central entity. It is possible for a domain controller to commit to a relation-ship of trust, effectively allowing the user of the other domain to gain access, but each user group from the other domain needs to be set up separately.

Dynamics: Although the design includes great flexibility in the number of user groups that can be created, the information stored in the registry is interpreted as data only. The registry does not provide a facility to insert a key such as "if these two keys exist, this key's value is true".

Scalability: The design of only one domain controller limits scalability, but the concept of inter-domain trust relationships facilitates it.

Administration: Administration is simplified by the possibility to create different groups, such as for backup or for local software installation tasks.

Least privilege: Although the possibility of creating fine-grained groups facilitates the principle of least privilege, common installations show that often it is required to set a key to "full control by everyone", because it is not clear which violation stopped the application from working correctly.

3.4.3 Certificate Based Access Control Frameworks

Certificate- (or credential-) based access control can be seen as a successor of capability-based designs. Instead of a system protected capability, a certificate with a well known format will be issued. Also, for gaining access, several credentials may be necessary, as opposed to only one capability providing access for one or more resources. Kerberos [23, 118, 83], an early but very influential system, was restricted to authentication. Sesame [26] was a European research project similar to Kerberos.

A paper first employing certificates in the full authorisation process was "Decentralized Trust Management" [20] by Blaze, Feigenbaum and Lacy. The certificates of their framework PolicyMaker consisted of programs written in a general programming language. Although this allowed great flexibility, the framework was hard to verify. Therefore, the authors proposed KeyNote [40], whose certificates consisted of a simple notation based on C-like and regular expressions. Another difference is the return value. KeyNote always returns a Boolean value, while PolicyMaker may return a set, which would satisfy the requirements for access.

Akenti [74, 159] is a recent framework sponsored and employed by the United States Department of Energy. Akenti employs three certificate types:

- X.509 user identity-certificates

- use-condition certificates
- attribute certificates

The identity certificates are generated and managed by certification authorities, such as Netscape CA or Verisign. LDAP[8] directory servers allow easy access, and verification is by SPKI[9]. Use-condition certificates are created by stakeholders, restricting the use of their resources. Attribute certificates are provided by attribute authorities attesting the user certain attributes. Both the use-condition and attribute certificates consist of a list of ASCII-keyword and value tuples that are signed by the issuer. An access cycle

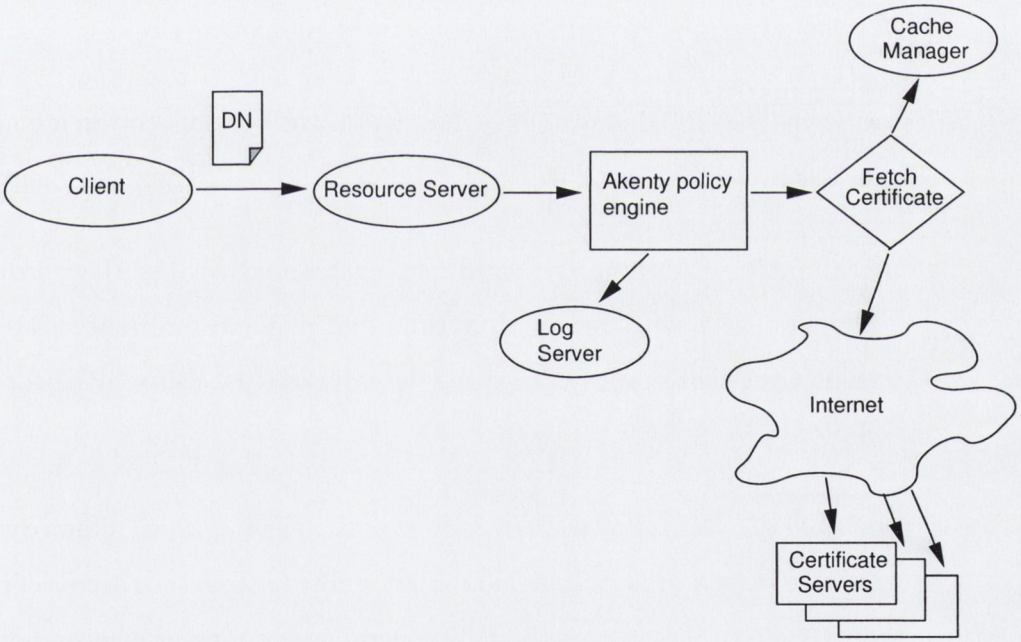


Fig. 3.11: Overview of the Akenti Architecture.

is shown in Figure 3.11(taken from [159]). The user provides an identity certificate to the resource server, which queries the Akenti policy engine. The engine may log the request for accounting purposes and fetch further certificates. Certificates may be

[8]Light Directory Access Protocol
[9]Simple Public Key Infrastructure

cached or fetched over WAN like the Internet. With each resource, an *authority file* is stored, which lists servers for identity and attribute certificates; the list of stakeholders; and servers that issue use-condition certificates for these stakeholders. Finally, all use-conditions have to be satisfied, which effectively allows each stakeholder the right of veto.

Assessment of Certificate Based Security

Extent of the TCB: Content of certificates are protected, which can also be policy code, as in PolicyMaker. Certificate based frameworks provide great flexibility for the protection of certain access-control components.

Locations of the TCB: By having the client merely fetch a certificate and providing the resource server with it, there is no need for a TCB on the client.

Decentralization: A list of required certificates and servers that provide these is enough for the collection of all information required for an access-request evaluation. Certificate providers can be decentralised by using SPKI for establishing relationships of trust.

Dynamics: PolicyMaker provides an active policy engine, which allows dynamic access-control models, while the engine cannot enforce mandatory access control. Akenti and KeyNote provide a subset of general programming environments and may not allow all dynamic policies.

Scalability: Certification-based frameworks have good scaling properties, because decentralised certificate providers can be employed. However, in Akenti, the policy engine fetches all certificates itself, which can introduce scaling problems when ther is a large number of clients and certificates.

Administration: The *authority files* of Akenti provide each server with the full power to decide which certificates it uses. The administration of the attribute and use-condition certificates is decentralised.

Least privilege: Certificate based access-control models are able to satisfy the principle of least privilege.

3.4.4 Hybrid Approaches

There are a few access-control frameworks designed with policy flexibility in mind. ARGOS [77] is motivated by integration of database systems into a federal database-management system. Access requests will be directed via the federal database-management system. Evaluation is done by a global access-control decision unit (the ARGOS framework), which is coupled with the local database systems. ARGOS utilises two paradigms: the owner paradigm and the administration paradigm. In discretionary-access-control (DAC) fashion, the owner paradigm allows the owner to decide who is allowed access. If the entry is owned by the special user *SYSTEM*, the administration paradigm allows an instantiation of a mandatory access-control model. However, once data are downloaded, further handling is at the discretion of the client.

Flask [154] aims to provide support for diverse security policies. Flask is based on a micro-kernal and employs for each access request an object manager, which queries a security server according to the security context of the requested object. Each object in Flask has two labels maintained by the security server. The first label is the *security context*, which is interpreted according to the security context. The second label is the security identifier, which allows the security server to classify the object to a certain security context. The object server is able to query the security server for decisions

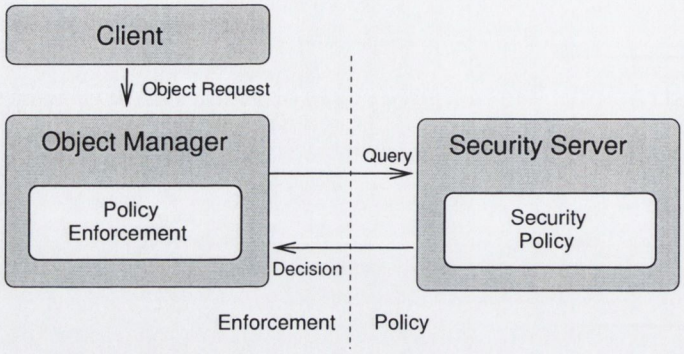


Fig. 3.12: The Flask Architecture.

regarding access, labelling, and polyinstantiation. The object manager also may cache the access decision to minimise the performance overhead. Finally, the object server can register for notifications if the security policies of objects in its governance change. Figure 3.12 [154] shows the enforcement policy abstraction of the Flask architecture. Hidden inside the security server is the security policy, which can be changed without incurring changes to the object manager.

Assessment of Hybrid Security

Extent of the TCB: Both architectures employ a special security component, which can be adapted to the needs of the access-control model.

Locations of the TCB: Because of their design goals (federal-database security or operation-system security), both architectures assume a global TCB.

Decentralization: Because of the assumptions of their designs, neither architecture facilitates decentralisation.

Dynamics: The policy engine of both architectures is hidden behind a uniform interface. This allows dynamic changes of the security model without reconfiguration of other architectural components.

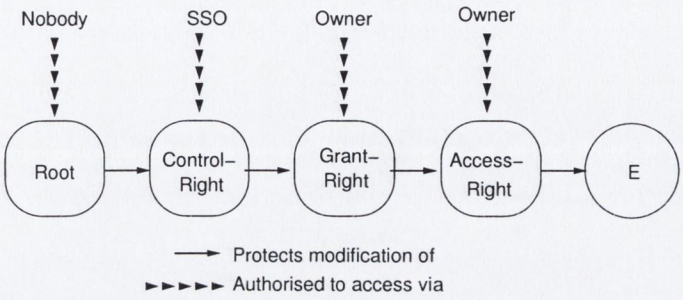


Fig. 3.13: ConSA Entity and Subject Labels.

Scalability: Because central TCBs are employed, both systems can face scaling problems.

Administration: Administration of the security models is done by directly implementing them into the policy engine.

Least privilege: There is no special precaution to facilitate the principle of least privilege, but it can be kept in both systems.

3.4.5 ConSA Security Module

With ConSA [125], Olivier proposes a security architecture external to the protected system. At the base of the ConSA system is the subject and entity (object) labelling. A minimal TCB will consist of protected labels that can be queried by the upper security modules. ConSA uses object-oriented concepts, which means that each label is an object itself, and as such, is protected by a label. Figure 3.13 [125] shows that an object (**E**ntity) is protected by an *access rights label*, the assess rights are protected by *grant rights*, and the *grant rights*, by a *root* label. The root label protects itself. The system security officer (SSO) has rights of control of all labels. He may permit other users to grant access rights to others.

Olivier shows how different security models can be instantiated by means of labels. Each security model will be implemented in an extra module on top of the labelling system. The labelling itself will be managed by the subject management module (SMM), which may be adapted for different labelling semantics.

Assessment of ConSA

Extent of the TCB: A minimal TCB will consist of the labelling scheme, while the concept of security components allows for the extension of the TCB to the needs of the upper access-control model.

Locations of the TCB: ConSA implies that the client interface will be directly operated by some user. Therefore, in cases in which the client interface resides in a security domain other than the server, the TCB will need to be extended to the client node.

Decentralization: The focus of the ConSA research was to show that different security models can be instantiated using a unified labelling scheme. Therefore, no decentralised mechanism was embedded into the design.

Dynamics: Dynamic security models are supported because the semantic of the labels is not restricted. A module instantiating a dynamic security model would be required to possess *grant rights.*

Scalability: ConSA is designed to replace the secuirty of the application or operating system, by providing a security component based on the underlying labelling system. The labelling approach can be very powerful, but it may also provide scaling problems. One example may be a security model, which requires a label to cover set of all subjects possessing access rights.

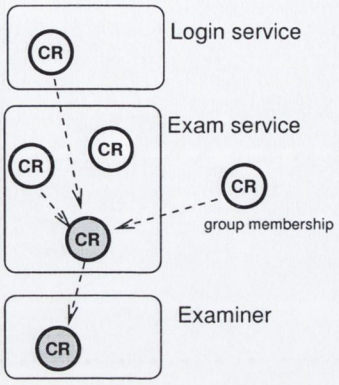


Fig. 3.14: A Credential Record Graph in OASIS.

Administration: Administration in ConSA is uniformly performed via the SMM module.

Least privilege: ConSA is orthogonal to the principle of least privilege. No special mechanism is implemented to facilitate it.

3.4.6 Open Architecture for Secure Interworking Services

Hayton [56] implemented OASIS[10] on the basis of role definition language. This work is interesting because it is a policy-based design. OASIS uses credential records (CRs) to represent knowledge or assertions about the client. Assertions can be role-definition language formulae or any other formatted data a service wishes to certify. A service can be a client to another service. This results in credential-record graphs. Figure 3.14 shows a credential-record graph. A log-in service testifies to an examen service that the user is logged in. The examen service provides the examiner with a credential record, that the user is at the moment taking the examen. Credential records from one service can be linked to external credential records of another service via the heartbeat

[10]Open Architecture for Secure Interworking Services

protocol. OASIS employs the heartbeat protocol to guarantee timely revocation and event notification. It was noted [57] that integration with other services using CRs is possible.

Assessment of OASIS

Extent of the TCB: OASIS credential records allow arbitrary content and are protected by the security architecture.

Locations of the TCB: Each service generates and verifies credential records relevant to its access-control decisions. A client may authenticate with a service but does not need to provide CRs directly. Therefore, no TCB is needed on the client side.

Decentralization: Each service can decide its own credential-record requirements which demonstrates decentralisation. Central services can be implemented using the heartbeat protocol.

Dynamics: Dynamic, system-wide updates can be distributed using the heartbeat protocol. Furthermore, each service is free to interpret and act upon the received credential records, allowing dynamic policy changes.

Scalability: A potential bottleneck can be the event-notification service (heartbeat protocol).

Administration: A role-definition language is used for flexible and unambiguous specification of access-control policies.

Least privilege: As OASIS is designed with role-based access control in mind, the principle of least privilege is likely to be maintained.

3.5 Summary

At the beginning of this chapter some criteria of access control models and mechanism were discussed. These criteria were then used to review different access control models. Our assessment of the different access control models revealed specific requirements for the access control mechanism. For example an access control model, which dynamically changes depending on the past access requests (c.f., Section 3.2.3), requires an active TCB.

Section 3.3 and 3.4 reviewed access mechanism approaches with respect to fitness for certain requirements. We are aware that, for example, a logic-influenced policy language requires active computing to derive the access control decision, but that the 'amount of activity' is not directly comparable with the one from the Chinese Wall security model. Also a hybrid access control mechanism may be able to instantiate all security models but does so by requiring a security officer to modify the mechanism source code.

Brief Comparison Using Starplots

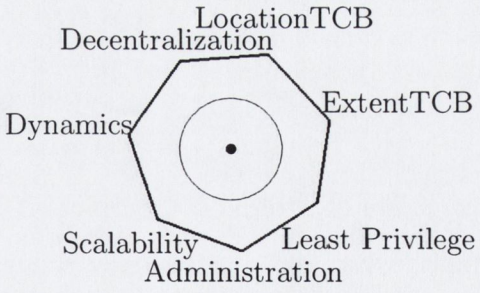


Fig. 3.15: Example Starplot.

To recap the meaning of the edges of the starplots. A value far away from the neutral circle means that the model has a high demand in this area or that the mechanism is able to support models with a high demand. The lower half of the plot consists of

desirable features for access control models. An ideal access control model starplot would be small on both sides. However an ideal mechanism would extend far on both sides. The example starplot shown in Figure 3.15 represents an ideal access control mechanism.

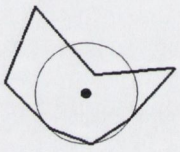


Fig. 3.16: Access Matrix

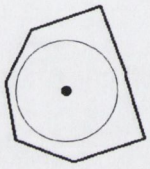


Fig. 3.17: Multilevel/-Mandatory

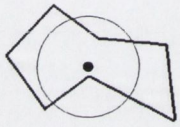


Fig. 3.18: Chinese Wall

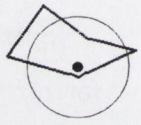


Fig. 3.19: RBAC

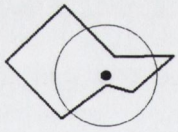


Fig. 3.20: Logic Influenced Policies

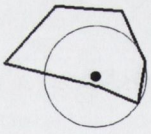


Fig. 3.21: Specificatin Lang

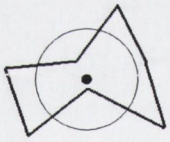


Fig. 3.22: Obligations

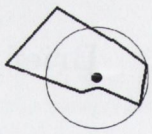


Fig. 3.23: Pol Objects

Comparing the access control model starplots it is seen that early models like the access matrix or multilevel model, posted high demands onto the access mechanism, but failed to take administration and least privilege into account. The Chinese Wall model also ignores the principle of least privilege, but provides good features of administration at the cost of dynamics and decentralisation. Notable is the similarity of the Chinese Wall starplot and the one of obligations. In fact, the Chinese Walls are obligatory to be built up after an access request in one conflict of interest domain. RBAC is interesting

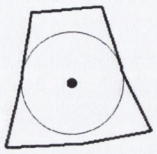


Fig. 3.24: Capability Based Mechanism

Fig. 3.25: ACL Windows NT

Fig. 3.26: Certificate Based

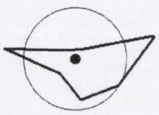


Fig. 3.27: Hybrid

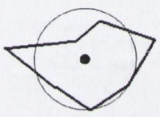


Fig. 3.28: ConSA

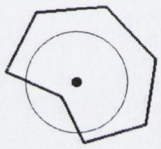


Fig. 3.29: OASIS

because it comes closest to the ideal access control model however it loses out on the dynamics side by providing administration features.

For the access control mechanism one can observe the progression of capability-based design which does not provide dynamics and extends the TCB to the client node, via the certificate-based mechanism to OASIS, which uses credential record design and also provides dynamics. The ACL implementation of Windows NT is the inverse of the ConSA mechanism, and the Chinese Wall access control model. Windows access control mechanism did not attempt to facilitate highly dynamic access control models, while ConSA was designed with the Chinese Wall model in mind. Finally one type of access control mechanism seems to be missing, which would be achieved by combining the scalability of the capability-based design, with the starplot achieved by the referral design of OASIS.

Chapter 4

Concepts Used in the Active Software Capability Framework

In the previous chapter we reviewed some access control mechanisms. We have shown that some designs are capable of instantiating one class of access control model but not others. The aim of the ASCap framework is to instantiate a wide range of access control models. Therefore, the design of the ASCap model is based on different elements of past research results. There are four different elements, which add independently and jointly to the flexibility of the access control mechanism:

- The ASCap Proxy
- The Policy Object
- The External Security Server
- The Active Software Capability (in short ASCap)

Most of the elements are based on concepts that already exist, but were used independently. We have derived a fifth concept, the security-flexibility-performance tri-

angle. This concept is more like a principle and can be used as a rule of a thumb to estimate whether a certain collections of requirements are feasible.

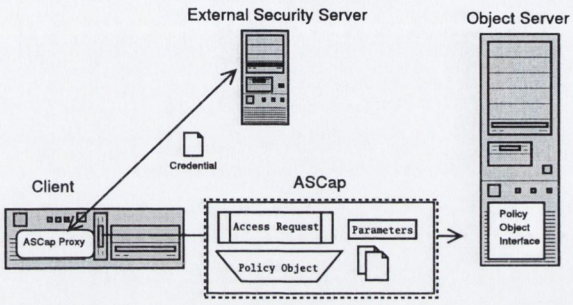


Fig. 4.1: Overview of the Framework Elements.

Figure 4.1 shows how the different elements interact. The ASCap proxy resides on the client and assembles the ASCap, which is sent to the object server. The ASCap includes the access request, policy object and credential(s), which are retrieved from external security server(s). The policy object will be executed on the object server (i.e. it acts like a server side proxy) and takes the credential and access request as input parameters. Additional elements of the framework, which do not belong to the core elements of the ASCap framework are the administration server or external rule servers and will be introduced later. Here we will present the origins and definition of each element together with our extensions. The aim of this chapter is to highlight the conclusions which we have drawn from past work on access control models and mechanisms.

This chapter continues with Section 4.1, which discusses the concepts underlying the ASCap proxy. In Section 4.2 the policy object will be presented and Section 4.3 discusses the origins and utilization of the external security servers. Then Section 4.4 introduces the format and aims of the ASCap[1]. Finally Section 4.5 presents the Security-Flexibility-Performance triangle, a principle aiding the design of actual access control setups.

[1] active software capability

4.1 The ASCap Proxy

The ASCap proxy is the client-side element of the ASCap access control mechanism. An ASCap proxy uses the proxy principle [150]: "*In order to use some service, a potential client must first acquire a proxy for this service; the proxy is the only visible interface to the service.*"

Shapiro proposed the proxy principle in the context of distributed system design. Originating in the area of programming paradigms, the idea of a proxy is to encapsulate the exact working schemes (in our case the access control model) of the service behind it. In the system of Shapiro, the TCB ensures that users can only generate proxies for objects they own and ensures that all communication other than local will be filtered by a proxy. Moreover, the TCB prevents the user from tampering with the proxy itself. These restrictions are too strong to hold in open distributed systems, but the concept of hiding the specific service interface behind the ASCap proxy adds to the framework's adaptability.

Neumann applies the concept of a proxy to the area of access control and adds to it the notion of restricted proxies [117]. According to Neumann a proxy delegates the full set of access rights of the proxy generator to the receiver, while a restricted proxy only includes a subset of these rights. The proxy in Neumann's framework is passed via the client to the end-server, similarly to the active capability concept above. We adapt the proxy principle and Neumann's idea of restricted proxy. In our design the ASCap proxy resides on the client side, and includes information on how to access the server for only the granted subset of actions. A further purpose of the ASCap proxy is to mediate between the client and additional security framework elements. The ASCap proxy does not act as an enforcement point as in Neumann's design, because open distributed systems do not provide a TCB on the client side, so integrity of a client proxy, cannot be assumed, and so the server must assume that it may have been modified in some way. In our design, in order to successfully access a server, a client

must present a valid policy object. Additionally, in many setups credentials from an external security server are required. The ASCap proxy will contact these external security servers and mediate between the users and servers to acquire the credentials. Unlike in Shapiro's design, no TCB is required to prevent the generation of arbitrary ASCap proxies, because possession of an ASCap proxy alone is not sufficient to gain access. In Section 5.7 we will discuss the security implications of this concept. Here it is sufficient to note that, like a restricted proxy, the ASCap proxy may deny service to the client, but cannot grant it on its own.

4.2 The Policy Object

The policy object is at the centre of the access control mechanism. It represents the access decision function and hence practically all of the access control model. However the full access control model behaviour will be influenced by another element, the external security servers, described later.

The policy object in the ASCap framework is a piece of mobile code, which can migrate to the server at access request time. There are two very similar concepts included in this design detail: (1) Access Control Programs; and (2) active capabilities.

1) Access control programs (ACP) were presented by Theimer et al. [120]. ACPs originally solved the problem of controlled right delegation, allowing users who want to delegate rights to untrustworthy objects to grant only a subset of their full privileges. Such users can write an ACP, which checks if the access request falls within the narrow window of delegated rights (principle of least privilege). Unlike a plain ACL system, where the server uses a fixed set of access rights after authentication of the client, ACPs are written by the delegator and are executed by the server, this allows generic restrictions to extend the fixed set of rights.

2) Active Capabilities, presented by Qian and Liao [131], replace the set of static

access rights in the access matrix by unforgeable scripts. These active capabilities are created and validated by the security manager. Qian and Liao chose a general purpose programming language as their scripting language, while for the policy object they use a component-based approach. Theimer et al. already noted that "The major concern with a general-purpose language is with the safety of the server." because a malicious client may write an ACP, which attacks the server. This concern does not apply to Qian and Liao's active capability system, as their security manager, which hands out the capabilities, resides in the same security domain as the one validating them. However in advanced setups of the ASCap framework parts of the policy object may originate from only partly trusted sources. Therefore we introduced rules as a component-based design of the policy object. Each policy object consists of several rules, whose single results will contribute to the final policy object decision. This concept allows for verifying rules which originate from partly trusted sources, using e.g. a sandbox environment.

Another issue is that a customised policy object will be used in each company. The security administrator who generates the policy object may not be an expert in programming code verification, hence providing off-the-shelf components that can be combined the actual security model simplifies his task.

4.3 The External Security Server

The elements presented so far have achieve great flexibility and provide access control models with an active, decentralised TCB. However some access control models require a centralised TCB, for example, to implement a revocation mechanism. Further, access control models which depend on the previous access history also require a method of recording and accessing this information. Without breaking the basic decentralisation of the ASCap framework the concept of security servers provides for the needs named

above. The external security servers provide input to the policy object by means of credentials, while the policy object itself is beyond reach for changes once downloaded by the client. The credentials may be freshly generated for each access request. Hence blocking a credential allows one to build a revocation mechanism.

The semantics of a credential are defined by the external security server and by the corresponding rule component, which interprets the credential. In general we understand credentials as data, although a specific implementation may use an execution environment inside the rule component to execute the credential. This does not break the ASCap design, but opposes our definition of external security server and external rule server. The external rule servers provide rule components, which will be an executable part of the policy object, while the external security servers provide credentials, which are data interpreted by rule components. This concept can be seen to originate in the Akenti framework [74] (c.f., Section 3.4.3). Unlike Akenti, which has one policy engine at its heart [159] that interprets all certificates, the ASCap framework relates each credential to a rule component, which will define its own semantics for this credential. In Akenti a missing condition certificate will yield additional access, the ASCap framework requires the administrator to specifically define a rule component for this behaviour.

The concept of external security server is also similar to Kerberos [118], where the ticket-granting server provides the client with a single certificate sufficient to gain access. In Kerberos acquiring a server ticket is a two-step process. The client first retrieves a long-term ticket granting tickets, which they can use to access the service tickets. Different security domains in Kerberos may commit to a trust relationship, which enables clients from one realm to exchange their ticket granting tickets for server tickets at the other realm. In all cases a single ticket (credential) is required to gain access. In the ASCap framework a single ticket may not be sufficient, but several may act together to form the system behaviour of the desired access control model.

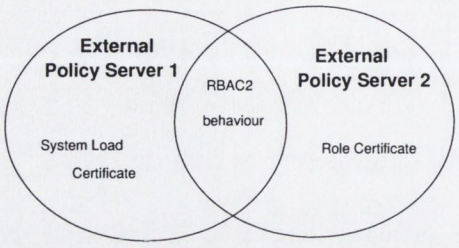


Fig. 4.2: Overall System Behaviour

Figure 4.2 shows two external security servers, each having a distinctive behaviour. The first provides a constraint based on the system load, the second provides a role certificate. Combining the relating two rule components into one policy object will result in a system with $RBAC2$ behaviour.

4.4 The Active Software Capability

The message, including the access request, policy object and parameters sent from the ASCap proxy to the server is called the active software capability (ASCap). The concept behind the ASCap is somewhat related to Jensen and Hagimont's "Protection Reconfiguration for Reusable Software" [72]. They describe a protection system based on hidden software capabilities [51]. Hidden software capabilities transform the general capability-based access control mechanism, which in early approaches [93] were mostly hardware implementations, to a pure software implementation. The resulting system of protection domains holding software capabilities [51] is extended with a separate reconfigurable protection system [72] for hiding the capability management from applications. Their prototype implementation was done in the ARIAS shared memory system. Unlike open distributed systems ARIAS provides a TCB on each network node. Our aim is to provide the same access control model transparency for the application programmer for open distributed systems. Jensen & Hagimant's concept of separate

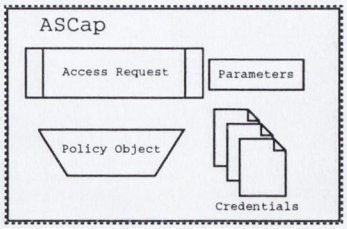


Fig. 4.3: The ASCap Template

protection domains could not directly be transferred, because there is no independent TCB on the client. The ASCap proxy in our design will provide the access control model transparency. The ASCap proxy is connected to the client application through a unified interface. Instead of employing an interface description language (IDL) to specify the current protection model setup, the ASCap proxy interface is fixed. The ASCap proxy will mediate between the application, the user and the different models: for example, the ASCap proxy manufactures the ASCap according to the specified protection model (i.e. policy object requirements).

Figure 4.3 shows the format of the ASCap. The format is similar to Li Gong's ICAP design (cf. Section 3.4.1), instead of embedding the identity into the random number, a short-lived authentication credential is included. This also allows for binding the capability to other properties than the identity. Finally to provide adaptability the policy object is included in the ASCap.

4.5 Security-Flexibility-Performance Triangle

Analysing different access control models and mechanism designs, we noted the interdependencies of different design (or later configuration) aspects. A highly secure access control model does not allow flexibility and puts restrictions on performance-enhancing decentralized elements. Figure 4.4 depicts these interdependencies. Each of these parameters can be seen as orthogonal to each other. A setup which is very flexible is

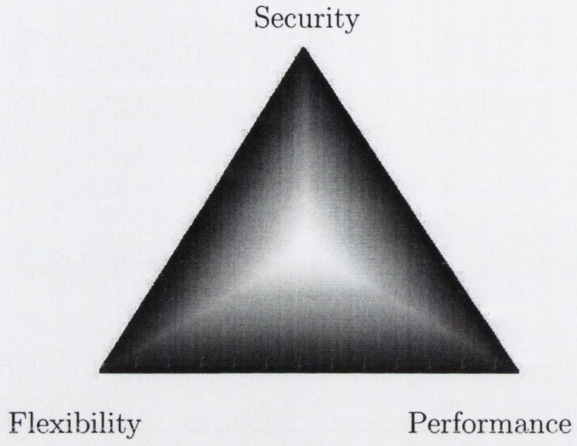


Fig. 4.4: Security-Flexibility-Performance Triangle.

hard to verify in terms of security and may include several indirect 'tweaking points' (e.g. external security server), which add to the performance overhead. Or a setup which has little performance overhead, such as one returning for all access decisions a 'granted' will be neither flexible nor highly secure. A highly secure setup should be fully verifiable, hence cannot be too flexible. If there is great flexibility the chance of unforeseen interactions (glitches) is higher and verification harder. Mandatory access control models, which are more secure against user errors compared to discretionary models, require a central server, which is a performance bottleneck. Given the fact that today's techniques of load balancing and redundant servers together with data replication can in fact cope with high loads, one might argue that high security access control with a central server does not come at cost of performance. However we believe the directional statement of the triangle is still correct. We will use the principle of security-flexibility-performance triangle throughout this thesis.

4.6 Summary

This chapter reviewed relevant literature with respect to the ASCap framework. The four elements of the ASCap framework have been discussed separately. For each element, the design concepts at its origin have been identified, and our own novel design choices have been stated. The final section introduced the security-flexibility-performance triangle, a principle derived from the past access control research. This principle not only influenced our design, but also allows for judging the feasibility of requirements for an access control mechanism.

Chapter 5

The ASCap Framework

After introducing the concepts of the ASCap framework in the previous chapter, this chapter presents the ASCap framework design. The design will be discussed from different points of view, such as architecture, usage, administration, and security. Each of these aspects is related to the aim of the ASCap framework to support all known access control models. Additional to the main aim, stated above, we discovered during the course of our research that the design also allows a novel joint access control administration. We have named the ability to delegate administrative power in a controlled manner: *partial outsourcing*. The last section of this chapter introduces the different outsourcing classes.

Section 5.1 provides a quick overview of the ASCap framework by explaining the two processes of the framework (initialisation phase and access request cycle). Subsequent sections set forth important points in various aspects of the ASCap framework. Section 5.2 introduces the architectural model, Section 5.3 the usage and programming model, Section 5.4 the administration model, and Section 5.5 the enforcement model. Section 5.6 highlights the use of PKI in the ASCap framework. Then Section 5.7 analyses the security of the framework and explains how a revocation, delegation or highly

secure setup can be instantiated. Finally Section 5.8 categorises classes of outsourcing access control and introduces the new paradigm of partial outsourcing. We conclude with a clear summary of the design features.

5.1 The Initialisation Phase and Access Cycle

This section introduces the full ASCap framework by stepping through an access request cycle in a coarse-grained manner. More aspects of the design will be discussed in detail in subsequent sections which examine the framework from different perspectives.

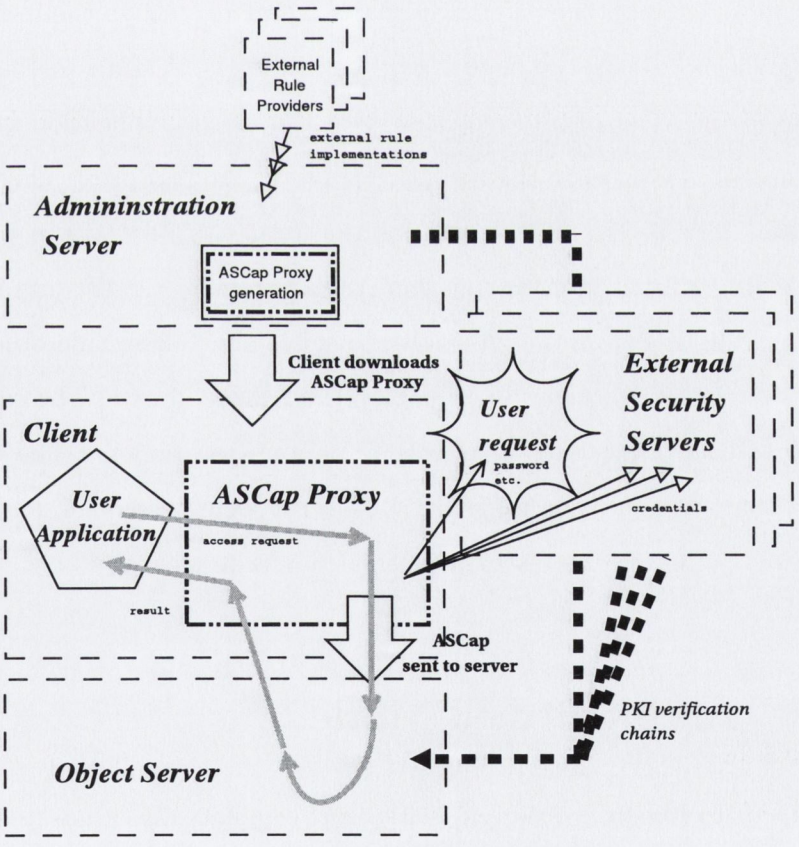


Fig. 5.1: Overview of the ASCap framework.

Figure 5.1 shows the full ASCap framework. Each logical part is shown in an

independent box. The administration server, client and object server are independent network nodes. The external security servers may also be independent network nodes or may be devices connected to one of the network nodes. The three main parts are in the centre and the external security servers are on their right. The external security servers are indirectly connected to the object server using a public key infrastructure (PKI).

There are two independent processes. First is an initialisation phase, which includes the ASCap proxy generation at the administration server and downloading of the ASCap proxy to the client. The second process is repeated for each access request. The application on the client hands the access request to the local ASCap proxy. The ASCap proxy sends the access request inside the ASCap to the object server which evaluates it. The result is delivered to the client application via the ASCap proxy. To successfully execute the access request the ASCap proxy may contact one or more external security servers, as well as the local user (e.g. for a password). There are two more external interfaces in the ASCap framework: the administration server can receive rule objects from external rule providers. These rule objects are incorporated into the policy object and ultimately the ASCap proxy. The second is found at the policy object which uses credentials as input parameters. The credentials can originate from external sources, namely external security servers, and influence the access decision similar to the server or client side parameters.

5.2 Architectural Model

The ASCap framework design splits some elements into separate servers: the administration server, for instance, is separated from the object server. This division allows for independent maintainance and administration of different functionalities of the system. The object server can be updated without reinitialising the administrative framework.

Likewise the access control administration can modify the security policies without modifying the object server. The architecture of the ASCap framework splits the (1) object server, (2) administration server, and (3) external security server into different design parts.

The object server has a specific function and therefore is separated from the administration elements.

The administration server is responsible for access control model adaptation, such as introducing new policies or revoking old ones.

External security servers are separated from the administration server, because their functionality is to provide and maintain all information required to run the access control model. There are other external security servers which are separated further because they belong to other functional domains such as IDS.

Any specific set-up may employ more than one external security server because different independent functionalities are required. An example is auditing and access control state maintenance of the Chinese Wall access control model. Splitting these parts to different servers allows each of them to be implemented in the most effective way.

Different architectural models are used for different functionalities. For example an auditing server may be replicated locally employing a hierarchical data collection mechanism to increase performance at event-logging time. Access control state maintenance cannot be implemented in the same way because information has to be added and retrieved frequently.

Another architectural design decision involved the ASCap proxy versus server proxy functionality. Although the framework does not demand that the ASCap proxy collects

the credentials from the external security servers, this does improves the framework's scaling and performance behaviour. Such a set-up also allows the client to cache credentials, or to deny access if not all credentials could be retrieved before contacting the server; hence decreasing the server's workload.

5.3 Usage and Programming Model

The usage and programming model is concerned with the application programmers (both client and server), as well as the administrator of the object server which uses the ASCap framework to govern access. The end user should not notice that the ASCap framework, instead of other hybrid approaches, was used to secure the application.

For the client application programmer, the ASCap framework provides a remote invocation or data query interface similar to middleware platforms. Likewise the client application has to provide an interface for the ASCap proxy. This interface is uniform for all access control models and has to allow the ASCap proxy to query the user for input and store semi-permanent data (e.g. cached credentials) on the local computer. The ASCap proxy is effectively a mobile code element that requires an execution environment. It is suggested that this execution environment is in the style of Java's sandbox. A sandbox provides the ability to restrict the mobile code's interaction with the client OS. It is also conceivable that the sandbox execution environment will provide only a subset of commands or use a purpose-built scripting language. The execution environment has to be defined by the ASCap framework standard to allow interoperation of different server and client application providers. Section 6.6 presents our choice for the ASCap framework prototype.

In all cases, the ASCap proxy provides, for each entry point of the managed service, a reference to the local client application. This reference can be specific, such as a stub with prototypes of method parameters, or general, employing a casting logic at the

server application side. In accordance with the concept of hidden software capabilities the initialisation of the ASCap proxy, for the client application, should not differ in any way from any general instantiation routine.

The server side application also has to provide an execution environment to evaluate the policy object, but because the policy object has no fixed programming language, there does not need to be a standardised execution environment. It is conceivable that policy object providers would deliver an execution environment for the server side with their objects.

The concept of hidden software capabilities promotes division of the application programming and security programming. The usage model reflects this in allowing the administration, which employs the ASCap framework, to employ rule components and policy objects from specialised access control programming companies. These "off-the-shelf" components can be customized by configuration files, which set only specific values like working hours, passwords, or addresses of external security servers. This allows software development project leaders to employ, for each programming part, the most cost-effective experts (application programmers, security programmers and administrators familiar with the local security policies) to customise the configuration files.

5.4 Administration Model

The administration model for the ASCap framework has to be able to take advantage of design features such as dynamic access control model changes and decentralized administration. Therefore the administration server is separated from the object server on its own network node. Although the object server always has the possibility to overrule a policy object, the administration server is understood to be the main point for administration changes. The instantiated access control model is determined by the

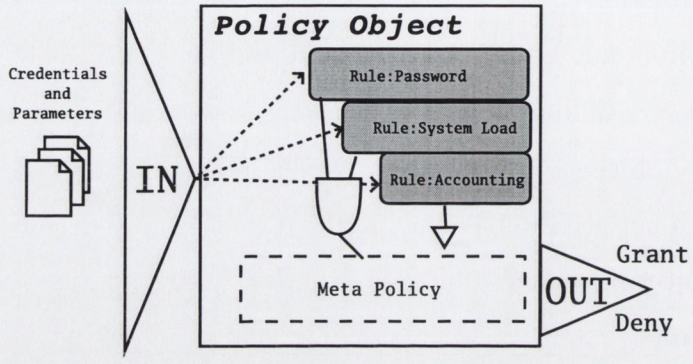


Fig. 5.2: The Policy Object Consists of Different Rule Components.

policy object, which is generated by the administration server. The component-based design allows the administrator to combine well-proven rule components to achieve the desired final access control model. A rule component can query the system (e.g. attached hardware) or user (e.g. password), query an external security server, or interact with the server application, to decide how to deal with the access request.

The administrator can introduce a rule object for each property, or policy, in case of a policy language driving the ASCap framework. Different rule objects are combined by a meta-policy to result in the final access decision (as shown in Figure 5.2). Our design is influenced in this element by Kuehnhauser's Meta- Policies [84, 86]. Kuehnhauser among others observed that in open distributed systems with multiple security policies, policy conflicts can arise [37, 15, 68]. During his research about user-defined security policies (see [85] for an introduction) Kuehnhauser develops a formal framework for supporting security policies in a multi-policy environment [87]. The framework allows reasoning about policy equivalence, policy cooperation and policy conflicts [84, 87]. Similar concepts can be seen in policy languages such as Ponder (cf. Section 3.3.2). Policy interaction is governed by policies about policies - named meta-policies. The ASCap framework incorporates meta-policies when instantiating

the policy object. Meta-policies govern the influence and interaction of rule components. Although it can be argued that conflict resolution belongs to the policy layer (cf. Section 3.3) we believe that the design of a unified access control mechanism has to reflect this methodology. If this were not the case, existing conflicts would be resolved by chance, such as in the decision regarding which policy is executed first. Another advantage of meta-policies at the access mechanism layer is that rule components (and hence external security server) can get only partial influence over the access decision. This allows, for example, the realisation of partial outsourcing (presented in the Section 5.8).

The administration of the ASCap framework is a two-step process, which integrates into best practice of current system design concepts. The top-down concept from software engineering [132] is applied to the security system design with the attack tree method [146]. Starting with the highest level objective, the analyst builds a tree of sub-objectives to the point of concrete system requirements. In the *first step* of our administration process the different requirements are translated into rule components. This can be done by choosing pre-defined rule components, employing an external security server with a related rule component, or by writing the rule in a scripting language. In this step each element of the access control model has to be represented by a rule component in conjunction with a configuration file, which holds the installation specific values (e.g. passwords). Additional rules may be added to include non-access control policies, such as accounting or administration obligations. Then in the *second step* the rule interactions are defined during the combination of rule components into the final policy object. This step includes choosing the sequence in which the rule components are queried, and making decisions that affect performance, such as whether querying an external security server can be done from the object server directly. The result of the administration process is a policy object (or a set thereof) representing the company's access control model.

After the administration has defined the policy object, the ASCap proxy can be derived. The behaviour of the ASCap proxy has to match that of the policy object, namely collect the correct credentials. The administration model allows the creation of an ASCap proxy which contains more than one policy object. This case provides the client with the freedom to choose the most suitable policy object for gaining access. For example an ASCap proxy can provide access to different entry points of the same server. If an entry point has more than one associated policy object, the ASCap proxy can check the local credential cache to determine whether one of the policies can be used without further client interactions - thus enhancing the system's performance. However, using more than one policy object in the ASCap proxy complicates verification of the system, and hence decreases overall security according to the security-flexibility-performance triangle principle.

5.5 The Enforcement Model

The enforcement model is influenced by the flexibility enhancing design concepts. The different elements of each access control model are enforced either by the policy object (evaluated on the object server) or by an external security server. In all cases the object server has the power to examine the elements of the policy object and check if the necessary credentials, as proof of enforcement, are present. The object server also has the power to reject policies issued by the administration server but, by doing so, interferes with the concept of free choice of access control model, as well as the administration model described below. While the object server has the final power over the policy enforcement, the administration server and the ASCap proxy have 'filtering power' on the access control model used. If the administration server does not insert the code regarding how to satisfy a certain rule, the ASCap proxy is unable to comply with the object server, which requires input for that rule. Likewise the ASCap proxy,

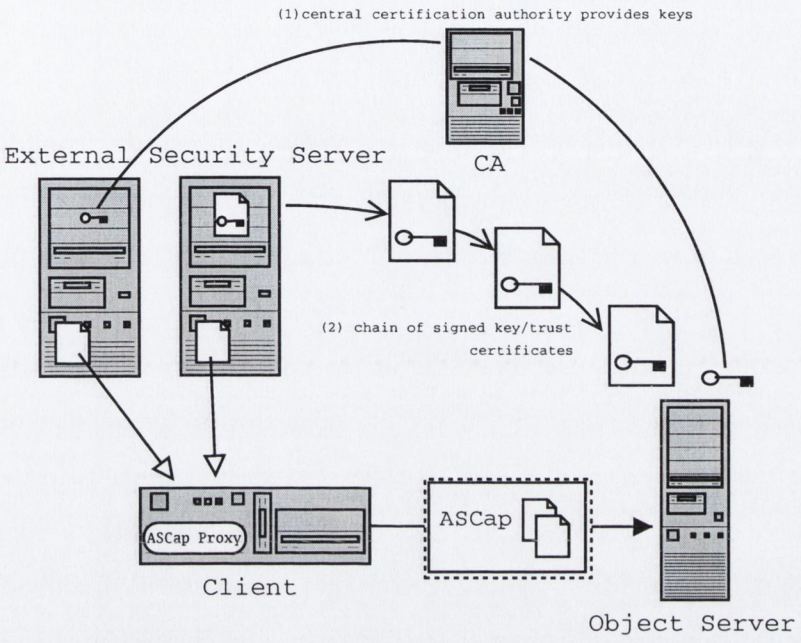


Fig. 5.3: Verification Chain of External Security Server Key.

which was modified by the client, may neglect to contact a certain external security server, which is required in order to satisfy one of the rule components. Therefore the administration server and an ASCap proxy modification can only limit the access rights, not extend them, if the object server does not comply.

As external security servers implement part of the access control model, they directly influence the behaviour of the whole system. It cannot be strictly enforced that an external security server implements the correct behaviour (only the credential format can be checked) but it is part of the trust relationship, analysed in the partial outsourcing section (cf. Section 5.8) below. However it can be controlled if the client has contacted the correct external security server and satisfied its requirements by checking the credential in the ASCap. The credential is signed using public key cryptography. Figure 5.3 shows the process of verification. Each private key of an external security server is linked to the object server via a certification chain. The concrete model used for verification such as (1) a centralised certification authority like in PKI [163] or (2)

an ad hoc chain as in the web of trust [80] is left to the implementation or employment case.

Being able to 'query' the same external security server from different object servers allows centralised access control models to be implemented. This enhances is the enforcement power of the *central* external security server. The object server only has limited options for monitoring correct access control enforcement, because it only observes the credential as the result of the enforcement. If the set-up requires a higher level of assurance, the ASCap proxy and rule component can be altered to have the client not contact the external security server, but either to provide the object server with all required information, so that it can compute the mandatory access control decision function itself including contacts to an external centralised database (effectively the same as an external security server). This set-up increases the workload of the object server from simple credential verification to active computation including network connections but may be done for the benefit of greater security. The trade-offs described above are in agreement with the security-flexibility-performance triangle principle.

5.6 The Role of the PKI in the ASCap Design

A PKI is used in two places in the ASCap design. Most prominent is for ensuring integrity and authenticity of the credentials delivered by the external security servers. The other is to ensure that the client did not modify the policy object. For both cases it is conceivable that different PKIs can be used, although this is not necessary. The PKI of the policy object is simple to establish because it is used by only the administration server and object server, which naturally are small in number and do not have complicated trust relationships (see 5.8 for the different trust relationships). The PKI for the external security servers is harder to establish because the trust relationship may

be more complicated. In Section 5.8 the different types of external security servers and trust relationships will be introduced, in this section we will review only the demands on the PKI.

The external security servers and object server may belong to different companies, which means to cooperate both would need to agree on one PKI root CA. Although the object server's company, as the customer, may want to operate the PKI, this is not most efficient, as then external security servers will need to use different private keys, depending on which company they serve at that moment. Similar, but less cumbersome is the case where each external security server operates its own PKI and therefore a company employing different external security servers, would need to participate in each of the separate PKIs. The ideal scenario is that the PKI is operated by an independent entity. This allows the external security server company to change certificates of external security servers (or add another server to the server farm) without the need to contact all customers. An independent PKI operator can ensures the object-server company that the external security server company does not arbitrarly generates new keys.

The external rule provider is a special case of the external security server. The rule provider sends executable rule objects instead of certificates. The rule objects will also be signed through the use of a PKI. In the ASCap framework the rule objects are received by the administration server rather then the client or object server directly. This allows the administration server to verify the rule objects not only by signature, but also using formal techniques (e.g. code verification, proof-carrying code, or model checking). These formal 'assurances' can replace a trust relationship established by a PKI.

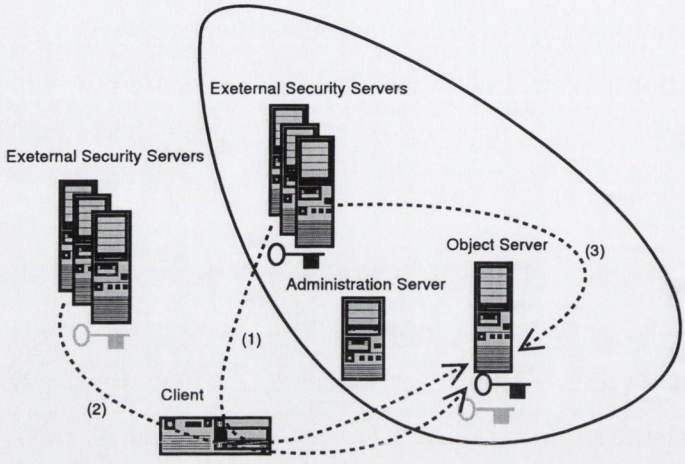


Fig. 5.4: Overview of the use of PKI.

Figure 5.4 shows the different places where the PKI is used. At (1) the credentials of an external security server residing in the same security domain as the object server are protected for integrity. Without a PKI a client could alter the certificates. However if the object server and these external security servers share a symmetric key, it is possible to ensure integrity without the use of a PKI. Case (2) shows credentials originating from an external security server outside of the security domain of the object server. This external security server is assumed to be out of the control of the customer company, while the one in case (1) was controlled by it. This difference is important once we use the approach of sharing one key among the object server and all external security servers. One external security server could generate a certificate impersonating any other of the group. This is not a problem in case (1), where administration is assumed to be friendly. In case (2) it is desired to enforce not only the integrity, but also authencity of the certificate and to be able to disable a single external security server by revoking his public key. Therefore a PKI cannot be replaced by a shared key approach[1]. In both cases it is not an option to send the certificate without in-

[1]We assume no approach where each external security server shares a separate key with the object server, because this would be similar to a real PKI.

tegrity protection, because it may be modified by the client. However if no PKI can be employed the ASCap framework can be configured so that the object server contacts the external security servers (internal or external), which does not require integrity protected certificates and is shown in case (3).

Finally in a real-world use the end-user wants to ensure that the ASCap proxy that will be executed on the client, does not harm the client. Java provides the method of sandboxing the ASCap proxy, it effectively runs in a restricted virtual machine. It is also possible to use code signing together with a PKI to establish trust into the ASCap proxy. However the ASCap proxy can also be implemented by the client and therefor would be known to be well behaving.

5.7 Security Analysis

All security relevant elements of the ASCap framework are included in the TCB. If one of these elements fails to provide the assumed service the full framework will fail.

5.7.1 The Object Server

The object server is part of the TCB. It executes the policy object, which is provided via the client after getting it from the administration server. The administration server is also part of the TCB. The client does not need to be part of the TCB because the policy object is cryptographically protected. In cases where external security servers are employed, each of them has to be trusted to some extent. External security servers do not always have to be part of the TCB because their credentials may not influence the access decision.

5.7.2 External Security Server

An external security server which provides, in the credential, not just a data value but an executable script, does have to be part of the TCB. This is because a screening of the executable script at the object server is not practical. Performance reasons will cause the executable scripts to be blindly executed. However, the framework supports rule component injection only from partially trusted external rule providers, as outlined in Section 5.8.

5.7.3 The ASCap Proxy

The ASCap proxy is executed on the client side. In terms of *client-side* security it has to be trusted. This trust relationship can be established by signing the ASCap proxy, as is done already with downloadable active web content (e.g. ActiveX, or Java Applets). For the *server-side* security the ASCap proxy does not need to be trusted. The purpose of the ASCap proxy is to mediate on the client side between the different external security servers, the user and the object server. The ASCap proxy on its own cannot grant arbitrary access rights. However it is possible that a Trojan Horse ASCap proxy could replicate the credentials a client rightfully acquired and reuse them for a different access request. This possibility is not outside the design assumptions, because each network node in an open distributed system has to guard its own perimeter. Client applications which do not want to allow ASCap proxies to be downloaded have to implement the functionality of all potential proxy configurations by their own means. For this situation the ASCap framework provides a description of the ASCap format which a client would have to obey. The description also covers how to retrieve credentials by running the required protocol with the relevant extended security servers.

5.7.4 The ASCap

The ASCap can be seen as the core of the TCB. The ASCap includes all required parts to evaluate the access request. Different elements of the ASCap, such as credentials or policy objects, are signed by active TCB parts to be protected against modification on the client side. The ASCap itself is signed and time-stamped to prevent replay attacks, but an attacker able to manipulate the ASCap freely can break the security of the framework.

5.7.5 Design of a Delegation Mechanism

Delegation in capability-based models is done by migrating the policy object (or full ASCap proxy) among the clients. In other access control model set-ups a delegation mechanism can be implemented by the use of external security servers. These external security servers will act as delegation servers, allowing registration requests to be handled, and referred to by a rule component. The policy object includes a rule that grants access if a valid credential from the delegation server is presented. In an online delegation service, the delegate registers with the delegation server at the same time as when the delegation request is issued. An offline delegation service requires the delegator to provide the delegates with an unforgeable token, which proves their identity to the delegation server.

5.7.6 Design of a Revocation Mechanism

Revocation can be implemented in the ASCap framework in two ways. A rule inside the policy object allows the object to expire causing the client to retrieve a new ASCap proxy. As the ASCap proxy can be distributed from different sources this mechanism allows decentralized revocation. The second implementation uses a central external security server which keeps a list of active policy objects. A client presenting this

server with a valid policy object receives a credential certifying that this policy object is still valid. This credential means, the object server does not perform revocation. The drawback of this mechanism is that it requires the external security server and object server to use synchronized clocks to prevent replay attacks.

5.7.7 Highly Secure Mandatory Access Control Models

One aim of the ASCap framework design has been to provide sufficient security to instantiate highly secure mandatory access control models. According to the security-flexibility-performance triangle principle, such a set-up comes at a high cost. We will review an example to highlight the reasons behind the preceding statement. Take the Bell LaPadula model (cf. Section 3.2.2) as an example of a high security mandatory access control model. This model specifies that no high-level subject (e.g. a process) can write to low-level objects. Lampson discussed the general confinement problem [90]. In the setting of open distributed systems, additional covert channels exist on the client node. By assumption (cf. Section 2.4) we do not presume to have control over the handling of data of a client-side application. An application may read (as a high-level user) top-secret data, then change identity and log-in as a low-level user, and write the data into a low-level object. One potential attempt may be to restrict the possibility of a client application changing identities, but as there is no TCB on the client side all such measures can be circumvented.

If someone requires a highly secure mandatory access control set-up, the ASCap framework adapts by the use of external security servers. An external security server can be part of the trusted computing base and hence prevent the migration of classified information. Such a set-up may provide the client with a secure information viewer, such as tamperproof, hardware-based code, as proposed by the trusted computing platform alliance, which is not running under the client's control. The system integrates this entity as an external security server. In an access cycle, the client credentials are

checked first; if the access is granted, the object server transfers the information to the secure viewing unit and upon termination receives back a credential, which may influence the final result sent back to the client. However, it is still possible for a malicious user to obtain a copy of the data by using a camera or pen and paper. Security can always be broken by means outside of the design specifications, which is shown in many examples by Schneier in [148]. Likewise a malicious client will be able to delegate access rights by providing a proxy service on his own network node which has been correctly granted access.

5.8 Partial Outsourcing of Access Control[2]

So far access control has only been seen as fully outsourceable. This section first analyses the different outsourcing classes (Section 5.8.1-5.8.4) and then presents in Section 5.8.5 the paradigm of partial outsourcing. The motivation behind this is the trend in today's business toward employing external consulting services for different highly specialised tasks [111]. The dynamics and economics of security outsourcing are clearly presented by Schneier [8, 148]. Allowing partial outsourcing of access control therefore contributes to the decentralisation of business structures of modern e-commerce ventures. We consider partial outsourcing to be a major contribution of this thesis and rooted in the design of the ASCap framework.

5.8.1 Class α: Single Administration, Internal

This class is understood as the base class, no outsourcing takes place and both administration and object server belong to the same domain. Figure 5.5 shows this scenario. The circle around the object server symbolises the security domain of a company. Inside this domain all parties are understood to behave with integrity toward the common security goals. Once a client retrieves an ASCap proxy and successfully accesses the

[2] **A version of this section was published in an earlier paper [4].**

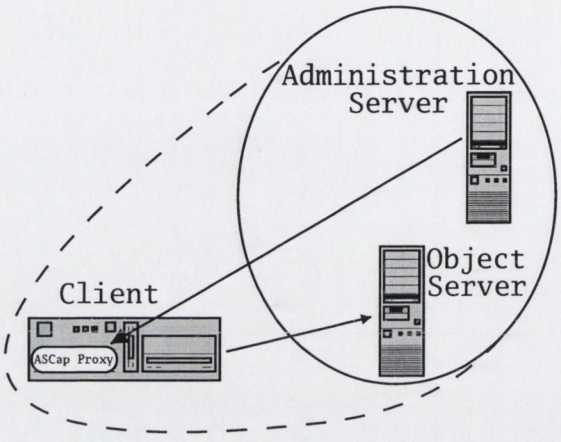


Fig. 5.5: Class α: No Outsourcing, Single Administration

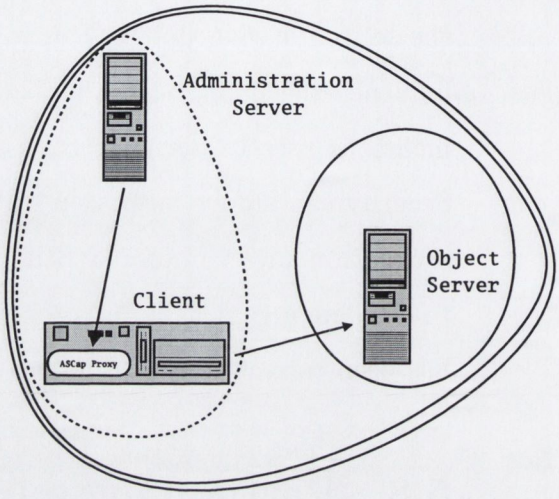


Fig. 5.6: Class β: Fully Administrated by External Company

object server, the dotted line shows that the client can be understood as obeying the same security goals.

Security analysis of this set-up is straightforward, because only one point of control exists. Authentication and authorisation of the client are fully handled by the company's own administration server.

Advantages are those of central administration known today.

Disadvantages include the fact that all the administration work has to be done by one party and no external expertise can be used.

5.8.2 Class β: Single Administration, External

Figure 5.6 shows the opposite extreme case: administration is fully outsourced. Again the circle around the object server symbolises the parts fully natively trusted. The

external administration server is not included in this domain because, by definition, the trust in this server is artificial. This time the dotted oval shows that the client belongs to the domain of the administration server, as it obeys his policies. Finally, the big double loop shows the necessary scope of required trust to make the system work. Only if both the administration server and client are trusted can an access cycle take place. The big loop can be reduced to administration server and object server if appropriate cryptographical protection is in place (e.g.signing and encrypting the credentials).

Security depends fully on the behaviour of the external administration server. In outsourcing terms this enables administration to benefit from the expertise of an external company, but also requires the external company to do less sophisticated but time consuming support tasks, such as createing user accounts and resetting passwords.

Advantage is that the full workload is outsourced.

Disadvantage is that the external company must be trusted.

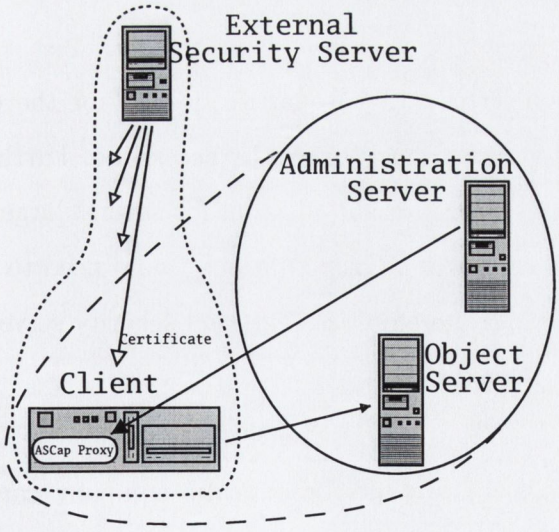


Fig. 5.7: Class γ: Outsourcing Using External Security Server Approach

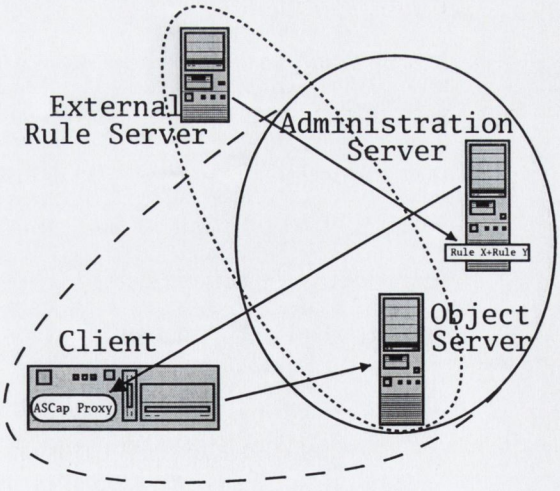


Fig. 5.8: Class δ: Partial Outsourcing Using an External Rule Server

5.8.3 Class γ: Outsourcing via External Security Server

In this class, as shown in Figure 5.7, administration is done by the company itself. Policies require the client to retrieve a credential of the external security server. This external security server is managed by the external consulting company. The multiple arrows hint that the same external security server might need to hand out credentials to a large number of different clients. Finally, the two dotted loops show that the client is dependent on the external security server but also has to play by the rules of the administration server. There is no big circle, which indicates that the external security server has no power over the object server, specifically it cannot give away access arbitrarily.

Evaluating *security* of this scenario shows that the company takes the ultimate decision of which security servers to ask, which demonstrates that the company has the most power over the access control. A malicious external security server can cause, at worst, a denial of service. The object server can protect itself from malicious external security server by code verification techniques [67].

A *weakness* of this set-up can be seen in the properties needed for the external security server. The external security server must always be accessible. Furthermore, imagining the scenario of a company doing external auditing for several large organisations, it would be the case that all clients of all organisations would need to retrieve a credential of the external company and therefore this external security server would need to deal with a large number of requests.

A *benefit* of this set-up is the possibility that the external company can decide to hand out only the credentials after different security checks have been successfully performed. Clients known to be malicious can be blocked although they may hold all other required credentials. Moreover as no executable code migrates from the external

company into the system, the extent of trust toward the external server is minimal. The system is slightly less flexible, however, because the number and address of external security servers and credentials will be fixed from the start of the system.

5.8.4 Class δ: Partial Outsourcing Using External Rule Servers

Figure 5.8 shows the final possibility. The external rule server delivers rule implementations to the local administration server. The administration server combines different rule implementations (here, Rule X+Rule Y) into policy objects and ASCap proxies (see Section 6.4 for details). The ASCap proxy is sent to the client. The dotted line around the client shows the known trust relationship. The second dotted line around the external rule server and object server indicates a natural trust relationship. This is so because the external rule implementations are executed by the object server.

For *security* the local administration has full control over which rule implementations are used. Because the external rule implementations are not the only ones included, the external rule server cannot arbitrarily permit access. However, there is a danger of the rule server providing a rule implementation which tries to break into the object server. This danger is much smaller than that posed by a malicious client and may be further reduced by code-checking techniques.

An *advantage*, as opposed to the previous class, is that the demands on the external rule servers are magnitudes smaller, because they interact with the administration servers and not each separate client. The flexibility of this approach is also easily demonstrated: the rule server may provide a rule, which calls an external security server, this class then emulates class γ.

5.8.5 Paradigm Shift: Partial Outsourcing

Today, we believe access control has been seen as only 'full' or 'not' outsourceable. This is represented by the classes α and β. Kerberos [118, 83] could be seen as implementing class γ. In this situation, the authorisation server could take the role of an external security server and the ticket-granting server would require ticket-granting tickets from more than one authorisation server. A simple realm change would instantiate only class β. This shows that, although Kerberos would be powerful enough for class γ, this method of access control outsourcing has not been realised.

We believe class δ has not yet been implemented. Allowing access control to be handled jointly by different parties would introduce additional possibilities. As a company is split into different departments, each rule of a policy could be motivated by a different need. For example, the human resource department could introduce checks to determine a person's affiliation, while the financial department might introduce rules about possible responsibilities of making payments. Each department would be able to define and modify its rules independently. Additional 'good practice' checks could be introduced by external companies, and none of the parties could cause a full disclosure on its own.

5.9 Summary

After a short introduction of the core process of the ASCap framework, we analysed the design from different perspectives. We presented the architectural model, enforcement model, usage/programming model, and a security analysis. The security relevant parts of the framework design were discussed and how to instantiate a revocation, delegation or high security set-up. Then we categorized levels of outsourcing access control, which led us to realise a new paradigm: *partial outsourcing*. The ASCap framework is the

first to support partial outsourcing. The benefits of partial outsourcing will be shown in the evaluation (cf. Section 7.5).

Assessment of the ASCap framework

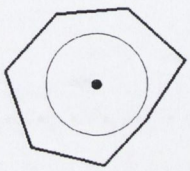


Fig. 5.9: Starplot of the ASCap framework.

To summarise the design features we will access the ASCap framework by the same criteria as used in the related work chapter. The starplot in Figure 5.9 shows that the benefits of a capability-based mechanism were combined with those of a certificate-based mechanism and further enhanced for a good scalabilty.

Extent of the TCB: The ASCap framework provides the policy object, credential and external security server as part of the TCB. This design allows it to support a wide range of access control models.

Locations of the TCB: The design of the ASCap framework specifically caters for open distributed systems by not assuming a TCB on the client node.

Decentralization: The use of external security servers in the capability-based ASCap design facilitates decentralisation. For example the external security servers can employ methods to provide a distributed centralised data storage, without the need of the rest of the architecture to be centralised.

Dynamics: The component-based design of the policy object allows instantiation of dynamic access control models, or a dynamically adapting access control model, depending on the environment.

Scalability: The ASCap framework does not rely on centralized entities. The ASCap proxy can be distributed from different servers only bound by a PKI. The external security servers provide the centralised data storage, such as the access history of the Chinese Wall model and the ASCap design can be implmented with today's replication techniques, which allow good scaling behaviour.

Administration: By separating the administration server from the enforcement architecture, administration enhancements can be designed without constraints from the access control mechanism. The paradigm of partial outsourcing facilitates joint administration and IT systems integration.

Least privilege: As the ASCap framework employs a capability-based design, the principle of least privilege is held. However the enforcement of the principle of least privilege is the responsibility of each access control model instantiation.

Chapter 6

A Prototype Implementation

In Chapter 3 we discussed different access control models and mechanisms. The last chapter presented the ASCap mechanism from different design perspectives. This chapter explains the realisation of this design in a unified access control framework that is implemented in Java.

Our implementation, as a prototype, does not provide an extensive user interface. We restrict our prototype application to a simple information reader and information provider service, but choosing a web server or database service is equally imaginable. Emphasis is put on the correct instantiation of different access control models without changing the core elements of the framework so that code changes are only necessary in the ASCap proxy at the administration server. A production quality implementation will most likely employ either one of the policy languages presented in Section 3.3 or a simplified scripting language to generate the policy object and ASCap proxy.

In the remainder of this chapter Section 6.1 provides an overview of both a full access request cycle and the class structure. We will discuss the implementation in a coarse-grained manner. This section should allow an understanding of the full ASCap framework. Then Section 6.2 outlines the programming environment used for

the prototype implementation. Section 6.3 examines the details of the administration server. We show how different example setups are implemented. The policy object class is discussed in Section 6.4. Section 6.5 presents an example of an external security server highlighting the features required to interact with the ASCap framework. Then Section 6.6 describes the ASCap proxy class and how it interacts with the client application and object server. Section 6.7 presents the client interface. Following this, the object server interface is presented. Finally Section 6.9 concludes this chapter with a summary of its content.

6.1 Overview

Figure 6.1 shows the full prototype implementation. The dashed boxes represent each autonomous network entity. At the top are one or more external rule providers. Then the three big boxes show the main parts of the ASCap framework, namely the administration server, the client and the object server. To the middle right the external security servers are shown. Also note the *user request* which has a similar interaction pattern as the external security servers.

Each arrow represents an object communication (the name of the transferred data is given by the label). The two big downward arrows each indicate a network communication. The first represents a communication of the administration server with the client, which downloads the ASCap proxy (shown in the dash-dotted box). Upon receiving the ASCap proxy the client has to verify its signature and a secure communication channel is set up with the object server. The second downward arrow represents the ASCap send from the client to the object server.

Finally the grey arrow chain shows an example access request. Although additional object interactions are necessary, we used grey only for the access request and successful feedback chain. It should be noted that each branch has to be processed (i.e. all

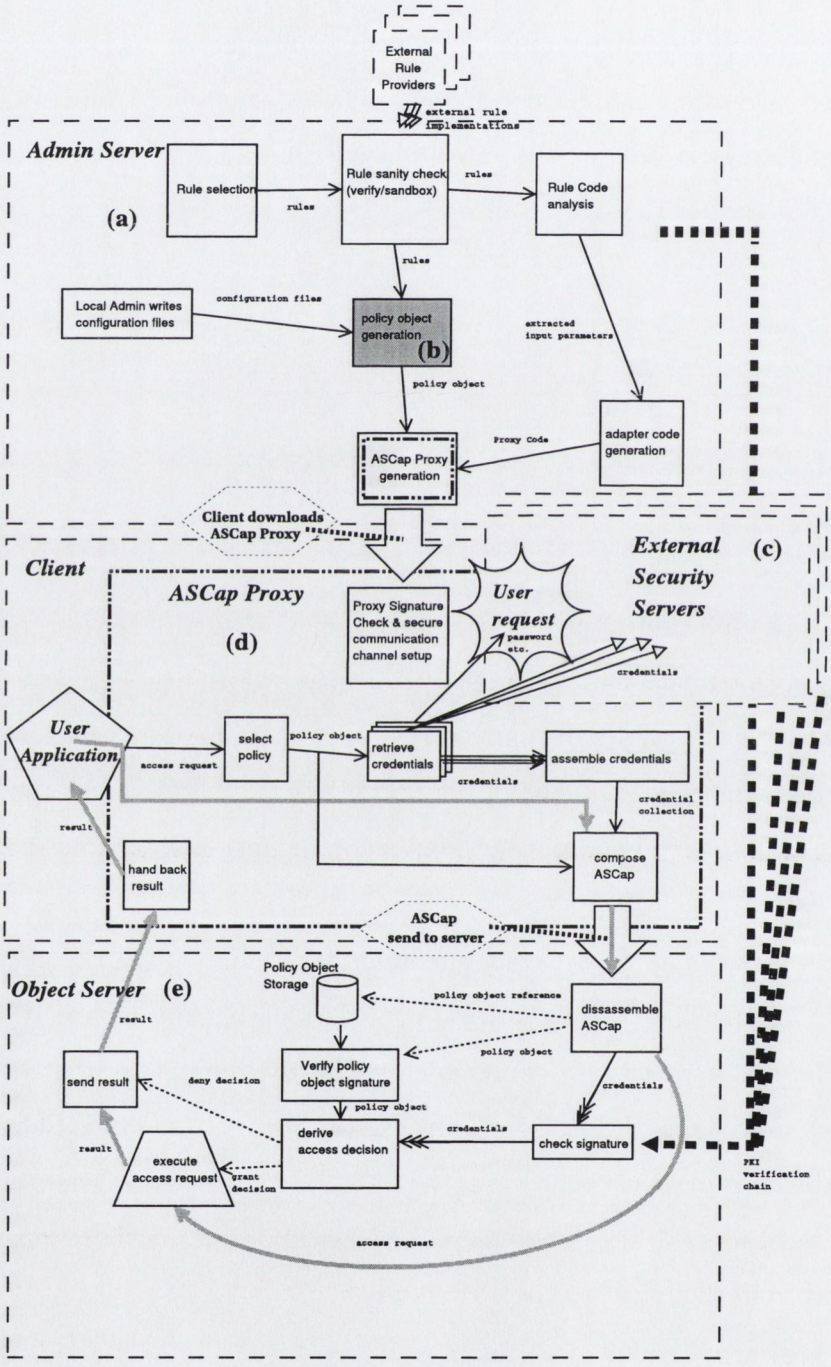


Fig. 6.1: Overview of the Full Prototype

credentials of the external security server collected) before the process can pass a box to which different arrows lead. In the object server there are two mutually exclusive branches shown by the dotted arrows. The first depends on the mode the framework operates in, whether the policy object implementation has to be retrieved from a local object storage or if it is send in the ASCap. In the second the processed branch depends on the result of the access decision function - either grant or deny lead to different results.

We will discuss the details of each box in the following sections, together with their implementation details, but first we will present the implementation environment.

6.2 Prototype Implementation Environment

Different approaches can be taken to implement the ASCap framework. A CORBA middleware-based approach hooks the access control framework between client and server stubs, while the mobile code aspect of the ASCap proxy would require the client side to download additional code. While CORBA itself provides platform and programming language transparency, the ASCap proxy may break this feature.

Therefore we also reviewed Java, which provides a platform independent programming environment. The JINI toolset provides a mobile code and service discovery API. JINI applications contact a look-up server to discover the required service. Each service provides a service proxy to the look-up server, which acts as a local mediator between the client and the service provider. All service interactions are done through the service proxy. This model is already similar to the access control programs and restricted proxy presented in the earlier Section 4.2. We therefore used the JINI framework to implement the ASCap framework. The prototype uses JINI Version 1.1; a short test showed that the current version 2.0.002 also works.

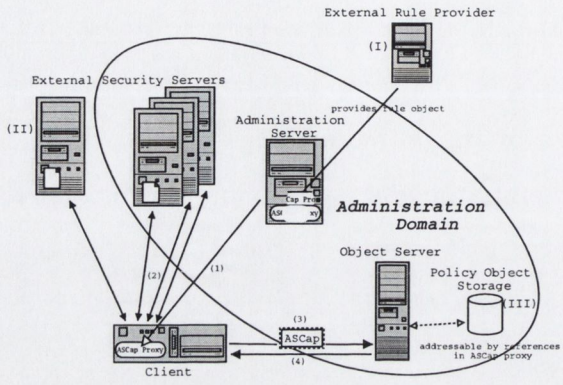


Fig. 6.2: Overview of the ASCap framework.

6.3 Administration Server

Figure 6.2 shows the full ASCap framework with respect to administration. The client application downloads an ASCap proxy from an administration server (1). The ASCap proxy includes the policy object (or reference) and adapter code to connect to various credential sources. The client itself is not part of the security domain (shown by the ellipse) and may be malicious, hence the policy object is protected by means of a cryptographic signature. The credentials can originate from different sources (2), such as the user or external security servers. Each retrieval might include extra communication steps symbolised by the double arrow. Then the ASCap proxy manufactures the ASCap and sends it to the server (3). The ASCap can be trusted, because all critical elements (policy object and credentials) are cryptographically protected. The server evaluates the ASCap and if access is granted returns the access request result(4).

Also shown in the figure are an external rule server (I) which may provide additional rule objects to be incorporated into the policy object. The external rule server and external security server (II) may also belong to a different administration domain than the object server. A variation of the framework allows the policy object to be stored on a secure storage (III) on the server side.

Implementation of the Administration Server

Figure 6.1(a) shows the behaviour of the administration server. The administration server gets input from three different sources. The rules selected by the local administrator, the rules provided by the external rule provider and the configuration files to adjust the rules to the local environment, (e.g. setting of working hours or disk quotas). The rules are checked for malicious behaviour by using code verification or a test environment. Each rule is analysed to derive the fitting ASCap proxy behaviour. The rules are combined to the policy object using the local configuration data. Finally the policy object, ASCap proxy adapter code are combined to generate the ASCap proxy, which is then signed to protect its integrity.

```
policies FirmIntern,FirmPublic,FirmPaying;                                    1

CA3 identified by public key {                                                3
"rO0ABXNyAC1jcnlwdGl4LnByb3ZpZGVyLmVsZ2FtYWwuQmFzZUVsR2FtYWxQdWJs\n"+        4
#some lines deleted                                                           5
"ADKDM1euEbgtlp8oEQYWtx4W37Empb9y186711pCZTclgBfUmV62p6koeBekbk0I\n"+        6
"krcX0A==\n"};                                                                7
RoleCredentialPayCust={"PayingCustomer" issued by CA3};                       8

#Policy = (rule1[,rule2,.];parameters1[,parameters2,.];command[,.]);          10
FirmIntern=("PersonalKey";"C@llC3nterPassw0rd";"Access");                     11
FirmPublic=("Load","minload=0 maxload=0.6","Access");                         12
FirmPaying=("RoleCert",RoleCredentialPayCust,"Access");                       13
```

Fig. 6.3: Example Configuration File.

Figure 6.3 shows an example configuration file. Line 1 creates three policies. Line 9 establishes trust into a role certificate issued by CA3, who's public key is given in lines

3-8; Finally in line 10 definition of the policies starts. Each policy consists of three parts:

rule: One or more rules are required for a policy to hold.

parameter: The parameter holds the custom values the security officer defines.

command: Command can determine the scope to which this policy is permitted to be used. In a more complex setup different access functions could exist e.g. if the company decides to introduce an advanced query service, which cannot be used by the public client. Then command would hold 'advanced access' and 'access' for the FirmPaying policy.

In the example scenario the *FirmIntern* policy only requires a static password ("C@llC3nterPassw0rd") to gain access. The *FirmPublic* policy requires the system load to be between 0 and 60% and the *FirmPaying* policy requires a certificate stating the bearer is a "PayingCustomer". This certificate can only be issued by the CA3 certificate authority.

The configuration file is used to manufacture the policy object, which is inside the ASCap proxy. In the prototype implementation the administration server uploads the ASCap proxy onto a JINI lookup server, which treats the ASCap proxy like any other service proxy. However for each access control model and object server combination there will be a separate service proxy.

6.4 Policy Object

The administrator assembles the policy object using provided rule components and configuration files to adapt to local parameters. This can either be done by a graphically enhanced administration tool that patches the rule classes which are already translated to binary format. Or an automated build environment that adjusts the local parameters

```
public class RuleTest implements Serializable {                          1
  public boolean test(byte[] creds, String rule_name, Object condition)  2
                                                    throws Exception{    3
    //creds holds the credential retrieved by the client                 4
    //rule_name indicates, which test to perform                         5
    //condition holds the local information, such as valid external      6
    //   server public keys and the roles, which yield a grant           7
    boolean result=false;                                                8
    if(rule_name.equals("PersonalKey")){                                 9
              String clientkey=new String(creds);                        10
              String servercopy=(String) condition;                      11
              if(clientkey.equals(servercopy)){                          12
              result=true;                                               13
    }         }                                                          14
    if(rule_name.equals("ActCap"))                                       15
    {// The client sends an object, which will be executed               16
        // credentials holds object with                                 17
        //      a) active capability                                     18
        //      b) input parameters for the active capability            19
        // conditions holds local input parameters                       20
        (...)                                                            21
        Object arglist[]= new Object[stor.arglist.length+1];             22
              arglist[0]= stor.arglist[0];     //from client             23
              arglist[1]= (String) condition; //from server              24
        result = meth.invoke(stor.activecap,arglist);                    25
    }                                                                    26
    if(rule_name.equals("RoleCert")){                                    27
        // Condition holds public keys of valid role servers             28
        //      and the role the client has to have.                     29
        rolecertclass conrc=(rolecertclass)condition;                    30
        //creds holds client rolecertclassrsserver                       31
        //   certificate and signature, encoded in a new class           32
        (...)                                                            33
        for(int mk=0; mk < conrc.ValidRoleServerPubKey.length;mk++)      34
        {// For each potential role server.                              35
              //load key and prepare algorithm                           36
              (...)                                                      37
              if(msigalg.verify(msig)) //test if signature is correct    38
                    tmpresult=true;                                      39
        }                                                                40
        if(tmpresult){                                                   41
              //Test credential role equals role in condition            42
              if(conrc.Role.equals(mycert.rolename))                     43
                    finresult=true;                                      44
        }                                                                45
   result=finresult;                                                     46
  }                                                                      47
  if(rule_name.equals("Load")){ ... }                                    48
  if(rule_name.equals("Time")){ ... }                                    49
}                                                                        50
```

Fig. 6.4: An Example Rule Object

in the source at compile time. Our prototype employs an ant [153] build environment and the local parameters are adjusted in *property* files. Ant provides a set of scripts for compiling and distributing Java applications. Ant also provides different tasks, not restricted to compiling, including file transfer, reconfiguration of local software (such as the web server) or calculating properties based on input parameters. While Ant scripts can be edited by the user, the general proceeding is to put modifiable parameters in property files keeping all user-editable parameters in one place. In the ASCap framework parameters like the look-up server address or port the server application listens on is set via the property files.

Figure 6.4 shows an example rule object. The different `if(rulename.equals("...")` blocks are the single rules, which are inserted into the rule object only if the corresponding property flag is set. Additional source code from external rule providers may be inserted. Each rule leaves the outcome in the variable *result*, which is handed back to the higher level meta-policy calling the single rules. The rule object provides a well-known method *test*, which takes *creds*, *rule_name* and *condition* as input parameters. *Creds* holds the credential provided by the client, which often originates from an external security server. *Rule_name* indicates the name of the rule to be checked, which is important if a rule object consists of several rules. Finally *condition* is provided by the server or policy object.

Figure 6.5 shows an example policy object of the

policy for financial clerks. The policy is RBAC-based and requires clients to present a valid role certificate (shown by *proofs[0]="RoleCert"* in the listing). The financial clerk can retrieve a certificate from one of two role servers (respectively linked to *ValidRoleServerPubKey[0]* and *ValidRoleServerPubKey[1]*). Additionally the server is going to check that it is inside normal business hours, which is determined by the server according to the "Time" rule with the times configured in the policy as input parameters. Finally the scope of the policy is defined, in our example, by a user

```
public class ExamplePolicies                                                    1
{                                                                               2
        public policyclass RBACFinancialClerk;// Financial Clerk                3

        public ExamplePolicies()                                                5
        {                                                                       6
         rolecertclass RoleServer;                                              7
         timeclass tmptime;                                                     8

         // A role server public key entry.                                     10

         // A role server entry, which has 2 public keys.                       12
         RoleServer= new rolecertclass();                                       13
         RoleServer.PubKeyHash=new int[2];                                      14
         RoleServer.PubKeyHash[1]=2844241;                                      15
         RoleServer.PubKeyHash[0]=3949657;                                      16
         RoleServer.ValidRoleServerPubKey=new String[2];                        17
         RoleServer.ValidRoleServerPubKey[1]=""+                                18
"rO0ABXNyAC1jcnlwdGl4LnByb3ZpZGVyLmVsZ2FtYWwuQmFzZUVsR2FtYWxQdWJs\n"+          19
(...)"xnx8Lg==\n";                                                              20
         RoleServer.ValidRoleServerPubKey[0]=""+                                21
"rO0ABXNyAC1jcnlwdGl4LnByb3ZpZGVyLmVsZ2FtYWwuQmFzZUVsR2FtYWxQdWJs\n"+          22
(...)"krcX0A==\n";                                                              23

         RBACFinancialClerk= new policyclass();                                 25
         RBACFinancialClerk.proofs=new String[2];                               26
         RBACFinancialClerk.reason=new Object[2];                               27

         RoleServer.Role="FinancialWorker";                                     29
         RBACFinancialClerk.proofs[0]="RoleCert";                               30
         RBACFinancialClerk.reason[0]=RoleServer;                               31

         tmptime=new timeclass();                                               33
         tmptime.aftertime=7; //working hours                                   34
         tmptime.beforetime=18;                                                 35

         RBACFinancialClerk.proofs[1]="Time";                                   37
         RBACFinancialClerk.reason[1]=tmptime;                                  38

         RBACFinancialClerk.commands=new String[2];                             40
         RBACFinancialClerk.commands[0]="Public";                               41
         RBACFinancialClerk.commands[1]="Financial";                            42
}       }                                                                       43
```

Fig. 6.5: An Example Policy Object

assuming the role *RBACFinancialClerk* to be allowed to execute commands of the class *public* and *financial.*

6.5 External Security Server

The design does not imply a specific implementation of the external security servers. The ASCap proxy communicates with an external security server via adapter code supplied by the provider of the external security server. We will give some examples of external security servers.

6.5.1 Role Server

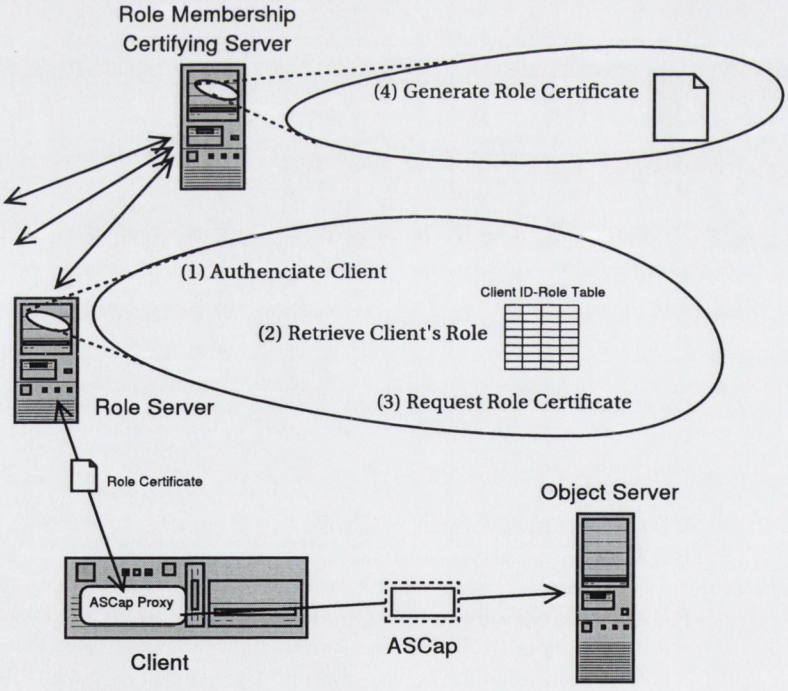


Fig. 6.6: Hierarchical Role Server Setup.

The previous section on the policy object used a role server. This section presents a

hierarchical role server setup to instantiate RBAC. The role server (or external security server) requires the client to authenticate. Internally a mapping table translates the client id to a role. In a simple setup the role server itself manufactures a role certificate, including a timestamp and signature. The setup shown in Figure 6.6 has better scaling properties. Here the role server does not have the power to manufacture a role certificate, but requests a certificate from a role membership certifying server. The role membership certifying server does not need to keep track of each client id, but only of authorised role servers and the different roles. The role certificate is delivered to the client which uses it to prove his role membership at the server.

6.5.2 Smartcard Based Security

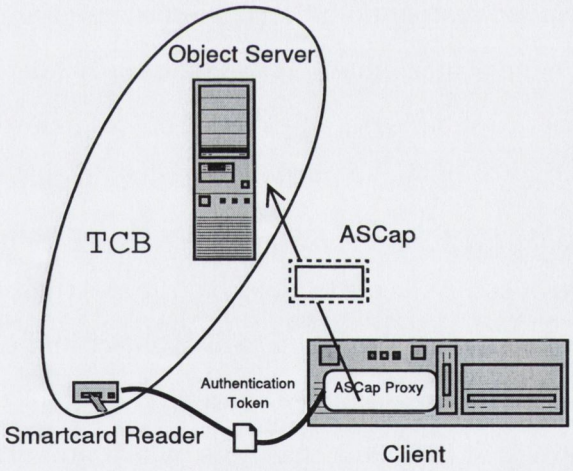


Fig. 6.7: A Smartcard Reader Integrated into the ASCap Framework.

Several access control frameworks proposed to use smartcards in access control [9, 66, 52]. These systems use the smartcard as a capability which the user possesses to gain access. Fig. 6.7 shows how the ASCap framework emulates this system by the use of external security servers. The smartcard reading device is seen as an external security server providing a TCB for the access control model. There are two possibilities: (1)

the ASCap proxy queries the smartcard hardware like any other local external security server, which provides an authentication token (or credential), as shown in the figure. Or (2) the smartcard hardware is connected via a secure line to the object server, in this case the rule object will query the smartcard hardware directly. In both cases it is easy to replace the smartcard technology by e.g. a biometric device, as the protocol between the external security server and (1) ASCap proxy or (2) rule object is not bound to the underlying hardware.

6.5.3 External Security Server as an Interface

In some cases it might be desirable to connect the access control framework to other systems, such as the intrusion detection system, in the network. There is no need to implement another custom-build interface, because the external security server interface provides enough flexibility. The ASCap proxy determines which information is provided to the external security server. For example the ASCap proxy may send a copy of the full access request to the external security server; or, in a different set-up, requires the ASCap proxy to send only the object server's address. In all cases the feedback is provided by the credential from the external security server. The credential is reviewed by the ASCap proxy and policy object and can have different influence on the access decision (c.f. Section 5.8). An alert state of the IDS may cause the access control framework to use additional tests, which are in normal cases regarded as too inconvenient and thus decrease productivity.

6.6 ASCap Proxy

Figure 6.1(d) shows the client node which downloads the ASCap proxy from the administration server. After checking the signature it instantiates the ASCap proxy which includes setting up a secure communication channel with the object server. The chan-

nel is bootstrapped by the use of public key cryptography. The ASCap proxy includes the public key of the object server which can be used to initialise a symmetric key cryptographic channel. Symmetric key cryptography is less expensive in computation terms. The channel setup includes server side authentication because only the object server is able to decrypt the symmetric key.

$$CL \rightarrow OS \ : \ \{CL, OS, Kclos, T\}Pkos \tag{6.1}$$
$$OS \rightarrow CL \ : \ \{data\}Kclos \tag{6.2}$$
$$CL \rightarrow OS \ : \ \{t, Command, Parameters, Credentials\}Kclos \tag{6.3}$$
$$OS \rightarrow CL \ : \ \{result\}Kclos \tag{6.4}$$

Fig. 6.8: Session Initiation Protocol

Figure 6.8 shows the protocol of the secure channel setup. The notation uses names for each message. The originator first and then the receiver second. In this example CL is the client and OS the object server. A curly bracket means information is bound to an object and either encrypted with the public key P_x, shared key K_y or signed with the private key S_z. $Nonce$ is a freshly generated random number.

Message 1 is encrypted with the pre-known public key of the object server. The public key is included in the ASCap proxy and protected by a signature to authenticate the object server for the client. $Kclos$ is a shared symmetric key and T a timestamp to prevent replay attacks. By this message the client can be sure, that it is talking to the object server corresponding to this ASCap proxy, while the server only knows it has a new session with an anonymous client.

Message 2 returns some initialisation data and completes the session setup handshake.

The following exchange is repeated for each access request and it can be seen, that

once a secure connection is established, the connection is stateless.

Message 3 represents all messages sent from client to the object server, which we name ASCap. All required information is included and the access decision is derived based on this information. The timestamp t prevents replay attacks of the ASCap. **Message 4** returns the result to the client.

The internal processes of the ASCap proxy are shown in figure 6.1(d). After the access request is given to the ASCap proxy, the adapter code retrieves all required credentials. Retrieving a credential may involve interaction with the user or interaction with an external security server. The credentials are assembled into a container object, which provides all required input parameters for the policy object (Figure 6.9 shows an example credential container for a role-based policy). The access request, policy object and credential container compose the ASCap, which is signed and sent to the server.

```
public class CertSigObj implements Serializable                    2
{                                                                   3
        public byte[] msig;                    //Certificate Signature   4
        public rolecertclassrsserver mcert; //Role Certificate      5
}                                                                   6
```

Fig. 6.9: An Example Credential Container

6.7 Client Interface

Because the prototype uses the JINI framework, the client needs to implement a JINI interface for service discovery. Figure 6.10 shows the client code, which queries the JINI look-up server and downloads the ASCap proxy (up to (...)). Then the ASCap

```
// Perform unicast lookup server discovery                                    1
LookupLocator lookup = new LookupLocator("jini://@lookupserveraddress@");     2
if(lookup==null){System.out.println("Lookup server not found !!!");}          3

// Get the service registrar object for the lookup service                    5
ServiceRegistrar registrar = lookup.getRegistrar();                           6
// and provide a template of the requested service                            7
ServiceTemplate template = new ServiceTemplate(null, null, serviceAttrs);     8

// lookup() returns the ASCap proxy for a service that matches                10
// the search criteria passed in as template                                  11
Object serviceObj = null;                                                     12
serviceObj= registrar.lookup(template);                                       13

//(...)  Omitted sanity checks and catching of exceptions.                    15

/* Session Initialisation phase */                                            17
((jab.informationprovider.InformationReaderService)serviceObj).init2();       18

/*An example access request */                                               20
result=((jab.informationprovider.InformationReaderService)serviceObj).        21
                                                callfunct2(myparameters);      22
```

Fig. 6.10: Downloading and Instantiating the ASCap Proxy

proxy is instantiated by calling the init2. For all communication with the object server the client application uses local ASCap proxy calls (callfunct2 in the listing), which will result in the generation and sending of the necessary ASCap. This allows the location of the object server to be hidden. Hence the ASCap framework could also be employed for operating system security, provided the client application has no way of circumventing the ASCap proxy interface. A second requirement of the client is that the ASCap proxy may require local storage for credential caching. Cacheing the credentials locally to the client does not pose any additional security threat, because the client is already in the system's RAM. Finally the ASCap proxy requires access to the user, such as the possibility of opening a password entry window.

It is conceivable for the prototype implementation to be written in native code which provides a JINI interface for its object server communication only. A different implementation may consist of a ASCap proxy interface employing a secure scripting language and a native code interface on the client application side. This setup would be similar to the CORBA framework, which provides stubs and ORBs for all programming languages.

6.8 Object Server

The object server receives the ASCap and disassembles it. The policy object can be of two types, depending on the mode the ASCap framework operates in (shown in figure 6.1(e) as dotted lines). In the 'active capability'- mode the policy object equals the policy object implementation, which can be used after signature verification. The second mode is less flexible, but provides a better performance. In this mode the policy object in the ASCap represents only a reference to local storage from which the policy object implementation is retrieved. Before executing the policy object each of the credential signatures have to be verified using public key certificates from a PKI.

The PKI information is either stored on the server or retrieved in real-time to facilitate a timely revocation service. The policy object represents the access decision function, which gets the credentials and access request as input parameters. Only if the policy object yields a 'grant' and the scope of the policy object shows that the access request is included, the object server executes the access request. The result is sent back to the client, which is either an access denied or the output of the access request.

6.9 Summary

This chapter presented the prototype implementation. After giving an overview of the implementation environment, the different elements were discussed. We showed a network view of the administration domain and the position of the different framework elements. The section on the policy object showed how single rules form the rule object and what the policy object looks like at source code level. The external security server section introduced different set-ups and how external security servers allow their implementation. It was shown that smartcard security, as well as interaction with other IT systems can be easily implemented. In the section about the ASCap proxy we discussed the requirements of the client application, including a comment on an alternative implementation. Finally a section described the internal behaviour of the object server.

Chapter 7

Evaluation

The previous chapters described the design and implementation of the ASCap framework. Different aspects and problems were presented, as well as the details of their solution. This chapter provides a comprehensive evaluation of all requirements and framework elements. First we will present the evaluation goals and outline evaluation strategies. Then we will proceed with a detailed evaluation.

The chapter continues as follows: Section 7.1 gives an overview of the evaluation chapter by presenting the goals and strategies. Then Section 7.2 presents the quantitative evaluation with sections on performance and scalability. In Section 7.3 qualitative properties, namely TCB on the client side, administration, adaptability and the unified interface, are discussed. Section 7.4 uses formal methods to prove the flexibility of the framework. Finally Section 7.5 evaluates the benefits of partial outsourcing by presenting novel intrusion detection strategies.

7.1 Evaluation Overview

In this section the goals are set and then the evaluation strategy is outlined.

7.1.1 Evaluation Goals

Arising from our requirements presented in Section 2.4 we identified the following evaluation goals:

Acceptable Performance Overhead The performance overhead experienced by the client application should not be overly expensive compared to the benefits of the ASCap framework.

Scalability The framework should provide good scalability similar to the Internet name service.

ASCap proxy is not part of the TCB: The framework should not require any trust in the ASCap proxy, which resides on the client. Elements like trusted hardware devices (e.g. a smartcard reader) provide access control models which require a TCB on the client side, without modifying the assumption of an untrusted client.

Flexibility The framework is flexible enough to instantiate a wide range of different access control models.

Administration Administration benefits from decentralisation by allowing each administration policy to be defined by the entity most suitable.

Adaptability The framework is able to evolve the access control policy without requiring changes to the client and object server-side applications.

Unified Client Interface By providing a unified security interface the client application can be developed independently from the access control architecture.

7.1.2 Evaluation Strategies

In the following we outline how each of the evaluation goals are treated, while the details of the arguments will follow in each section.

Acceptable Performance Overhead

We will present measurements of the performance overhead and compare these with well-known security mechanisms like SSL or the Akenti [159] framework. This comparison allows us to put the performance overhead into perspective.

Scalability

Measurements of the scalability when several external security server are employed will be presented. It will be shown how scalability improving techniques used in the Internet name service can be employed to provide good scalability for the ASCap framework.

ASCap proxy is not part of the TCB

We will discuss the attack tree of a client account compromise and show that the influence of a trusted ASCap proxy is negligible. Then we will introduce a trusted external security server (e.g. smartcard reader) and show how security will be improved.

Flexibility

An informal evaluation of the flexibility is achieved by showing how several influential access control models can be instantiated. We will present a standard ACL set-up, a sparse capability-based system, the extended identity-based capability system and an RBAC system. A formal proof that a large class of access control models can be instantiated correctly will be given in the formal evaluation Section 7.4.

Administration

We will review the paradigm of partial outsourcing and how it enables decentralised administration. The evaluation also includes an identification of the ASCap framework elements that enable partial outsourcing.

Adabtability

We evaluate the adaptability of the ASCap framework by showing how the framework is able to dynamically change the security policy enforcement mechanism from a Kerberos system to an RBAC-based system.

Unified Client Interface

We will review the client interface interactions of different access control models to show that the ASCap framework provides a unified interface.

7.2 Quantative Evaluation[1]

The quantitative evaluation consists of a performance and scalability evaluation. We measured the performance of the prototype and compared it to SSL and other performance results found in the literature. The scalability evaluation shows how the mechanism of the ASCap framework enables shifting the workload from the server to the client, allowing better scalability.

7.2.1 Performance of the Prototype

We have measured the cost of instantiating the framework and initialising an active software capability in order to evaluate the overhead imposed by the framework.

The measurements of Table 7.1 were made on a Pentium III 300, 128MB Ram, Linux 2.2.18, JVM 1.3.1-b24 mixed mode. The times are in milliseconds. We measured the time at the client application, which corresponds to the time taken from the call to the ASCap proxy until the result of the access request is delivered back to the application. The 1st access column shows loading of the ASCap proxy relevant classes and including

[1] **A version of this section was published in an earlier paper [5].**

context switching. Subsequent accesses measure the real access request time required by the framework.

Operation	1st access (ms)	subsequent accesses (ms)
Null invocation	1043	339
SSL	-	372
$RBAC_3$ 1 credential	1091	413
$RBAC_3$ 5 credentials	1230	522
$RBAC_3$ 10 credentials	1324	614
$RBAC_2$ (system load at server)	1419	560

Table 7.1: Measurements of Different ASCap Setups

The first measurement (Null Invocation) compares the performance of the basic framework to SSL. In this set-up no access control is done. After session initialisation, our framework is approximately 10% faster than a normal SSL connection. This is mainly attributed to the fact that we use AES [29] instead of Blowfish [145], which is used by SSL.

A session initialisation of 1043ms seems expensive, but considering that the experiment was performed on a fairly slow machine and involves an authentication protocol which includes generation of a symmetric key, it becomes more reasonable. Generation of the session-key requires sampling of randomness to be used by the cryptographic library, this takes the majority of the time[2]. The prototype is implemented in Java, which has a reputation for being slow, e.g. the performance measurement of the Akenti framework [159] using a Java SSL enabled client accessing an Akenti/Apache Web server names an average total access request time (Akenti on top of SSL channel, 1KB file) of 762ms if caching is used, without caching this increases to 2.96s. We argue that the performance of the ASCap framework is comparable to other Java-based security

[2]A test performed using the cryptographic algorithm of java 1.5.0beta resulted in only half the session set-up time. However it is not clear how much gain has to be attributed to the use of a faster CPU Athlon XP 2.6+.

solutions and that the performance overhead incurred using the ASCap framework is acceptable.

7.2.2 Scalability

There are three distinct issues of scalability:

- Scalability of the credentials
- Scalability of the external security servers
- Scalability of the number of clients

Scalability of the Credentials

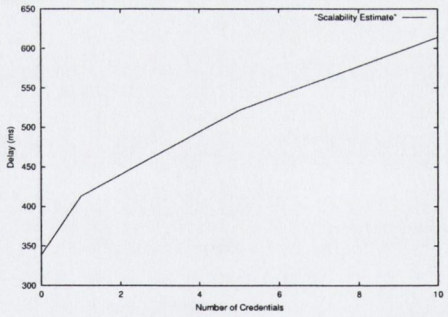


Fig. 7.1: Scaling of Number of Credentials.

Figure 7.1 is generated using the values of Table 7.1 and shows the scalability of credentials. It can be noted, while the number of security servers doubles with each step, the latency increases only by a fraction yielding a good scaling behaviour toward the number of credentials. Although this set-up emulates security models which include many external security servers, we would expect only a small number of credentials to be used in real life situations.

Scalability of External Security Servers

The last line of Table 7.1 measures the influence of the behaviour of the external security server on the latency imposed. Although only two credentials are used, the one with the system load delays the access request significantly, this suggestes that the scalability of the external security servers is important. The ASCap framework does not pose any restrictions on the external security servers other than the result has to be handed back in form of a credential, therefore, all scalability enhancing can be used. For example the Internet naming service employs a hierarchical structure exposing only a few servers to the large number of clients. A similar approach would allow the external security servers to hide a synchronisation infrastructure behind a few well-known servers.

Scalability of the Number of Clients

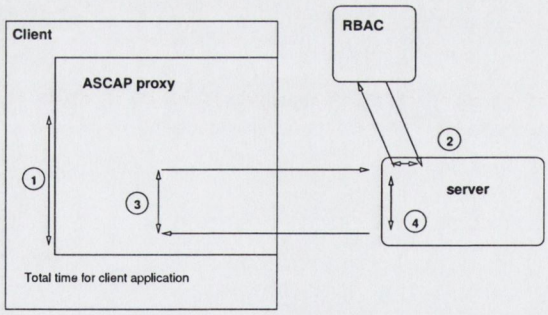


Fig. 7.2: Traditional RBAC setup

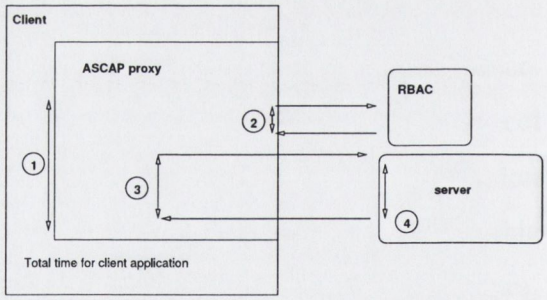


Fig. 7.3: ASCap RBAC setup

Scenario	1)Total time	2)RBAC call	3)client call	4) server side
ASCap (Fig. 7.3)	440	37	160	91
Traditional (Fig. 7.2)	214	36	211	164

Table 7.2: Times Measured for Figures 7.2 & 7.3

As the number of clients increases, the object server, executing the policy object, poses a bottleneck. Therefore, the ASCap design provides a mechanism to offload work from the object server to the clients. As shown in Figure 7.3 the ASCap proxy will contact the external security server instead of the object server (Figure 7.2) as in traditional set-ups. For example, a traditional RBAC set-up requires a form of database or storage of role memberships at the server. A client will only need to send its authenticated identity and the access control decision is performed entirely by the server. For comparison reasons we have stored the same information in an external RBAC server. In our ASCap client-based setup, the client retrieves the role membership certificate, packs it into the ASCap object and sends it to the server. The server in this scenario will only verify the credentials.

The total time seen by the client application increases by more than 50% compared to the traditional set-up. This is due to an unfortunate effect of Java serialisation/de--serialisation that we are currently working on eliminating[3]. In a working environment, the client will cache the received role membership certificates and on each subsequent request gain 51ms. This means that the cost of retrieving, decoding and re-encoding the membership certificate is amortised if the client invokes the server 5 times with the same role membership certificate.

As expected, the RBAC server call takes the same time (36ms) regardless of who is doing it, while the work on the server increases by 40% (91ms simple verification compared to 128ms (164ms-36ms) retrieval and verification) in the traditional set-up compared to the ASCap set-up. The benefit of the ASCap client-based solution is that in large scale systems, the workload and resource usage is distributed over the clients. Typical bottlenecks, such as the number of network sockets at the object server are avoided, while scalability improving features, such as a larger number of external security servers, can still be employed.

[3]Using the ByteBuffer of the java.nio.* package introduced in java 1.4 only about a 20th of the time is needed, which demonstrates the costs of a less clean implementation.

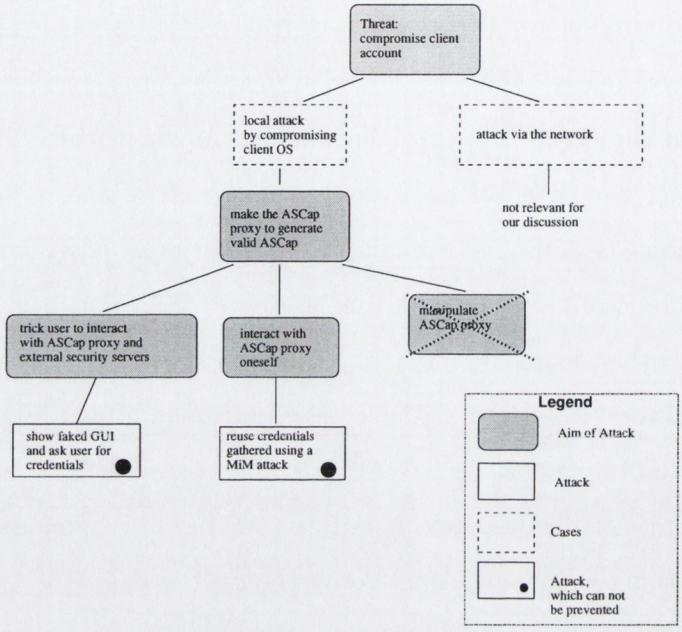


Fig. 7.4: Attack Tree of Client Account Compromise.

7.3 Qualitative Evaluation

This qualitative evaluation section discusses properties which cannot be directly measured, such as the effect of a TCB being present on the client side; the flexibility, shown by presenting different set-ups; the administration; adaptability and unified client interface; and later, in Section 7.4, formal methods.

7.3.1 ASCap Proxy not Part of the TCB

Figure 7.4 shows the attack tree for a client account compromise. We have only depicted the part relevant to show that a trusted ASCap proxy does not increase the system's security. Therefore we have omitted, for example, the branch of non-local attacks, such as network-based attacks. A full attack tree would show several attacks, such as the replay attack or modifying credentials, which are prevented by the ASCap design.

We assume that the attacker was able to compromise the client OS and therefore

does not rely on network-based attacks. The attack crossed out by the dotted line (manipulate ASCap proxy) would be the one prevented if the ASCap proxy is part of a TCB. However, there are other options available. Having control over the client OS allows the unpreventable attack of the user being conned into giving away the credentials. Or, during a legitimate user interaction, the credentials can be recorded and reused silently in later interactions with the ASCap proxy. It is worth noting that, in both attacks, the threat is against the credentials held by the user (e.g. passwords). Credentials, which are retrieved by external security servers are timestamped and are protected against modification. Therefore, an external security server, local to the client node, improves the security significantly.

External Security Server Based Security

An external security server, such as a smartcard reader, connected to the local client PC (cf. Section 6.5.2), provides the level of assurance needed by security models which rely on the client-side integrity. A smartcard reader provides a TCB for the user interface, not for the ASCap proxy, which otherwise would be handled by the compromised OS. External security servers can also provide assurance. In cases where information flow should be controlled on the client side, then an external security server could act as an information viewing unit.

7.3.2 Flexibility

In this section we will show informally how different access control models can be instantiated using the ASCap framework. In Section 7.4 the same will be done formally. It is important to note that in all scenarios the object server does not know the access control model in advance, but learns it by receiving the policy object together with the ASCap.

ACL

In the ACL model, the ACL has to be stored in a protected area. This can be done in three ways: (1) store it on the object server, as in the traditional set-up; (2) store it inside the policy object; or (3) store it on an external security server. In this setup we will use the third version.

Prior to sending the ASCap the client sends a request for the relevant ACL entries to the external security server which hands them back in a protected credential. The client sends this ACL excerpt together with authentication tokens to the object server, which derives access acording to the ACL model.

Sparse Capability

According to the sparse capability model possession of the capability is sufficient to gain access. This can be achieved by storing a long-lived credential on the client node. The ASCap proxy retrieves this credential and includes it in the ASCap. Therefore, the use of the ASCap as an element that includes all necessary elements to derive access, is sufficient to instantiate capability-based access control models.

Identity Based Capability

In identity-based capability models, the capability includes an authentication token. We implement this by providing an authentication server (external security server), which provides a (second) credential. The long-lived client capability credential together with the client ID credential form the full identity-based capability system.

RBAC

We have presented set-ups instantiating the RBAC model before (for example, in Section 6.5). The RBAC set-up employs several external security servers whose internal behaviours correspond to the relevant models.

7.3.3 Administration

The administration model (cf. Section 5.4) of the ASCap framework allows the access control policy to be split into different rule objects. Each rule object can originate from a different source. Administration can be simplified by allowing each rule of the access control policy to be defined by the most suitable entity. For example, the financial department of a company knows best about spending regulations and are up-to-date about changes. Allowing the financial department to introduce a spending rule into the policy object, therefore, not only simplifies administration but also increases security. However the financial department may not be fit to decide whether the client is properly authenticated, therefore this rule has to be written by the security officer. Partial outsourcing supports jointly administrating the access control policies. Without partial outsourcing, the financial department would provide the security officer with the information about the spending policies and would in most cases leave the security officer with the task of translating these policies into correct rule objects. Furthermore, the security officer would need to act for all policy updates. While using partial outsourcing the security officer only needs to act if rule interdependencies change (e.g. a spending check points out that the amount is negligible and therefore no further authentication effort should be done).

Partial outsourcing not only provides the benefit of distributing the access control administration internally, but also the advantages of enabling external entities to be included in the access decision. The different classes of access control outsourcing (cf. Section 5.8) enable adjustment to attain the level of assurance and flexibility required. Some administration benefits from partial outsourcing can be seen by the examples of new intrusion detection strategies presented in Section 7.5.

7.3.4 Adaptability

Adaptability can be achieved by supporting dynamic policy changes. The ASCap framework supports dynamic policy changes through the use of the ASCap proxy and external security servers.

A given policy can be revoked using the revocation mechanism outlined in Section 5.7.6. This causes the client application to download a new ASCap proxy, which instantiates the new access control policy. This mechanism allows a security policy based on identity, which may be enforced by Kerberos (c.f. Section 4.3), to be replaced with a role-based access control mechanism. The new ASCap proxy will automatically contact the new set of external security servers in order to retrieve the appropriate credentials.

Minor modifications are supported by the use of external security servers alone. For example, an identity-based external security server, which provides a permission ticket after the client has authenticated, may restrict the client's permissions to certain times in the day by not issuing tickets for that client outside those hours or by providing a rule in the policy object that denies access to objects, even if the client has a valid permission ticket. In the first case, we either have to modify the external security server itself or the policies that governs the issuing of permissions. In the second case, we simply have to modify the rule that is stored in the policy object.

7.3.5 Unified Client Interface

In Section 6.7 the client interface was presented. This is indeed a unified client interface, because the source code shown in Figure 6.10 does not need to be changed regardless of which access control model is instantiated. This allows the application programmer to only implement the unified client interface, while the access control specific changes are done in the ASCap proxy code.

Although the access control model changes can be hidden from the application using this set-up, they cannot be completely hidden from the user. For example, in one set-up the user is required to input a password, while in another he is not. Therefore the ASCap proxy requires the ability to query the user or store credential records on the local filesystem.

7.4 Formal Evaluation[4]

In Section 7.3.2 the flexibility of the framework has been shown using example setups of different access control models. In this section we will evaluate this property formally. First, it will be shown how the π-calculus can be used to express access control models with the example of the ACL model. Further more this example is used to present a validation technique and how to reason about behavioural equivalence. The following section presents the process expressions of two versions of the ASCap framework and the traditional setup. Both versions of the ASCap framework will be shown to be weak late congruent to the traditional setup. In the final section we will discuss these results.

7.4.1 An Access Control Model Expressed in the π-Calculus

In this section we will demonstrate that the π-calculus is able to express access control models with the example of a simple access matrix model. To enhance the practical relevance we will start with an informal description of the model, draw some conclusions to understand the model and derive the process expressions in the last step.

Informal Policy

In a company, a user might be given numerical user-ids instead of usernames. The users are divided into three groups with different permissions. The informal policy,

[4] **A version of this section was published in an earlier paper [3]**

formulated by the security administrator, reads as follows:
Odd numbered users only have r rights, even numbered user have rw rights, if the user-id is divisible by four the user gets rwx rights. Our system only uses the accounts with id 1 to 5.

Access Matrix

From the informal policy above an access matrix as described by Bell [12] can be built. Each row summarises the rights a user has, while each column shows all permissions to the corresponding object. Intersecting cells hold the specific access of the user to the object. Below an access matrix of the example policy is shown:

User	Object 1	...	Object n
1	r		r
2	rw		rw
3	r		r
4	rwx		rwx
5	r		r

Process Expression Version 1

To derive the process expression for the given policy, one may first determine the row (user-id) and then the access rights. This would result in the process expression below:

$$
\begin{aligned}
P_{acl1} &\stackrel{def}{=} l(u).[u=1]U_1([u=2]U_2([u=3]U_3([u=4]U_4([u=5]U_5 D)))) \\
U_1 = U_3 = U_5 &\stackrel{def}{=} l(p).[p=r]GD \\
U_2 &\stackrel{def}{=} l(p).[p=r]G([p=w]GD) \\
U_4 &\stackrel{def}{=} l(p).[p=r]G([p=w]G([p=x]GD)) \\
G &\stackrel{def}{=} \bar{l}grant \\
D &\stackrel{def}{=} \bar{l}deny + l().D
\end{aligned}
$$

This process receives information on the channel l, which can be understood as a login channel. After some transitions the result (grant or deny) is transmitted on the l channel. P_{acl1} is the start state of the policy process, U_1 etc are the states according to the user-id, G can be understood as a Grant state, and D as deny. In D, the process may receive additional information on $l()$, which will be ignored. This behaviour is required as an invalid user-id might result in a default denial regardless of the right requested.

Process Expression Version 2

A different implementation might derive the process expression directly from the informal description, thus resulting in the following process:

$$
\begin{aligned}
P_{acl2} &\stackrel{def}{=} (\nu k)(S_{accountcheck} | S_{rightcheck}) \\
S_{accountcheck} &\stackrel{def}{=} l(u).[u = 1]\overline{k}o([u = 2]\overline{k}en4(\\
&\quad [u = 3]\overline{k}o([u = 4]\overline{k}e4([u = 5]\overline{k}oD)))) \\
S_{rightcheck} &\stackrel{def}{=} k(g).[g = o]U_o(\\
&\quad [g = en4]U_{en4}([g = e4]U_{e4}D)) \\
U_o &\stackrel{def}{=} l(p).[p = r]GD \\
U_{en4} &\stackrel{def}{=} l(p).[p = r]G([p = w]GD) \\
U_{e4} &\stackrel{def}{=} l(p).[p = r]G([p = w]G([p = x]GD)) \\
G &\stackrel{def}{=} \overline{l}grant \\
D &\stackrel{def}{=} \overline{l}deny + l().D
\end{aligned}
$$

Note that the indices refer to the following meanings:o= odd (i.e., in respect to an imaginary user-id), en4=even, but not divisible by 4, e4=even and divisible by 4. $S_{accountcheck}$ can be understood as translating a user-id into a group (g)(or role like in RBAC [41]), while $S_{rightcheck}$ decides the permissions.

Validating the Process Expressions

For validation, rule-test expressions can be written, which are a complement to correct policy implementation. The rule-test sends out a $\overline{passed}$ to the environment upon completion to indicate correct policy behaviour. Some rule-test expressions are given below:

$$
\begin{aligned}
R_1 &\stackrel{def}{=} \overline{l}1.\overline{l}r.l(z).[z = grant]\overline{passed} \\
R_2 &\stackrel{def}{=} \overline{l}1.\overline{l}w.l(z).[z = deny]\overline{passed} \\
R_3 &\stackrel{def}{=} \overline{l}6.\overline{l}r.l(z).[z = deny]\overline{passed}
\end{aligned}
$$

Note, that it can be shown that all possible combinations (rule-test,policy expression) will reduce correctly, but for space reasons we will print out only the following two systems:

1. $T_1 = P_{acl1}|R_1 \rightarrow^* \overline{passed}$

2. $T_2 = P_{acl2}|R_3 \rightarrow^* \overline{passed}$

Case 1) $T_1 = P_{acl1}|R_1 \rightarrow^* \overline{passed}$

In the following we give the reductions used.

$$l(u).[u = 1]U_1([u = 2]U_2(\quad | \quad \bar{l}1.\bar{l}r.l(z).[z = grant]\overline{passed}$$
$$[u = 3]U_3([u = 4]U_4([u = 5]U_5 D))))$$

REACT (Communication on l).

$$[1 = 1]U_1([1 = 2]U_2(\quad | \quad \bar{l}r.l(z).[z = grant]\overline{passed}$$
$$[1 = 3]U_3([1 = 4]U_4([1 = 5]U_5 D))))$$

MISM1 and expand U_1.

$$l(p).[p = r]GD \quad | \quad \bar{l}r.l(z).[z = grant]\overline{passed}$$

REACT (Communication on l).

$$[r = r]GD \quad | \quad l(z).[z = grant]\overline{passed}$$

MISM1 and expand G.

$$\bar{l}grant \quad | \quad l(z).[z = grant]\overline{passed}$$

REACT (Communication on l).

$$0 \quad | \quad [grant = grant]\overline{passed}$$

Ignoring empty process and MISM1.

$\overline{passed}$ $\square$

Case 2) $T_2 = P_{acl2}|R_3 \rightarrow^* \overline{passed}$

Reduction rules from the appendix A.1 are given for each step.

$$(\nu k)(S_{accountcheck}|S_{rightcheck}) \quad | \quad \bar{l}6.\bar{l}r.l(z).[z = deny]\overline{passed}$$

Expanding $S_{accountcheck}$ and $S_{rightcheck}$.

$$(\nu k)(l(u).[u = 1]\bar{k}o([u = 2]\bar{k}en4(\quad | \quad \bar{l}6.\bar{l}r.l(z).[z = deny]\overline{passed}$$
$$[u = 3]\bar{k}o([u = 4]\bar{k}e4([u = 5]\bar{k}oD))))|$$
$$k(g).[g = o]U_o([g = en4]U_{en4}([g = e4]U_{e4}D)))_k$$

REACT (Communication on l).

$$(\nu k)([6 = 1]\bar{k}o([6 = 2]\bar{k}en4(\quad | \quad \bar{l}r.l(z).[z = deny]\overline{passed}$$
$$[6 = 3]\bar{k}o([6 = 4]\bar{k}e4([6 = 5]\bar{k}oD))))|$$
$$k(g).[g = o]U_o([g = en4]U_{en4}([g = e4]U_{e4}D)))_k$$

MISM2.

$$(\nu k)([6 = 2]\bar{k}en4(\quad | \quad \bar{l}r.l(z).[z = deny]\overline{passed}$$
$$[6 = 3]\bar{k}o([6 = 4]\bar{k}e4([6 = 5]\bar{k}oD))))|$$
$$k(g).[g = o]U_o([g = en4]U_{en4}([g = e4]U_{e4}D)))_k$$

4x MISM2 and expand D.

$$(\nu k)(\bar{l}deny + l.D|k(g). \quad | \quad \bar{l}r.l(z).[z = deny]\overline{passed}$$
$$[g = o]U_o([g = en4]U_{en4}([g = e4]U_{e4}D)))_k$$

REACT (Communication on l) and expand D.

$$(\nu k)(\bar{l}deny + l.D|k(g). \quad | \quad l(z).[z = deny]\overline{passed}$$
$$[g = o]U_o([g = en4]U_{en4}([g = e4]U_{e4}D)))_k$$

REACT (Communication on l).

$$(\nu k)(0|k(g).[g = o]U_o([g = en4] \quad | \quad [deny = deny]\overline{passed}$$
$$U_{en4}([g = e4]U_{e4}D)))_k$$

MISM1.

$$(\nu k)(0|k(g).[g = o]U_o([g = en4] \quad | \quad \overline{passed}$$
$$U_{en4}([g = e4]U_{e4}D)))_k$$

Because k is the only reduction possible, but k is not known outside of the scope, the process and restriction can be ignored. This argument goes along RES4 of the proof system.

$$\overline{passed} \quad \square$$

Showing Behavioural Equivalence

Validation as done in the previous section may show similar behaviour in the context of certain test expressions. But for achieving confidence that both expressions correctly depict the informal policy it is necessary to show that both processes are weakly late bisimilar.

First, we have to notice that version 2 consists of two concurrent processes communicating by the bound channel k. $S_{rightcheck}$ has no other means to be reduced then by initially receiving on k. $S_{accountcheck}$ final action is sending on k. If we are able to convince ourselves, that $\tau_{[u=1]}$ in $S_{accountcheck}$ (see Section 7.4.1) strictly leads to $\tau_o \stackrel{def}{=} \bar{k}o|k(g).[g = o]$ (see Figure 7.6), we can show that $P_{acl1} \approx P_{acl2}$ using the

bisimulation S.

$$
\begin{aligned}
S \;=\; & ((P_{acl1}, P_{acl2}), (P'_{acl1}, P'_{acl2}), (U_1, U_o), (U_1, U''_o), (U_3, U_o), (U_3, U''_o) \\
& , (U_5, U_o), (U_5, U''_o), (U_2, U_{en4}), (U_2, U''_{en4}), (U_4, U_{e4}), (U_4, U''_{e4}), (U'_1, U'_o), \\
& (U'_3, U'_o), (U'_5, U'_o), (U'_2, U'_{en4}), (U'_4, U'_{e4}), (D, D), (G, G)).
\end{aligned}
$$

Using the tree diagrams given below, it is possible to verify the correctness of the bisimulation. Please note that leaves, which result in the same state, such as D or U_o are only printed once, but abbreviated for a better overview.

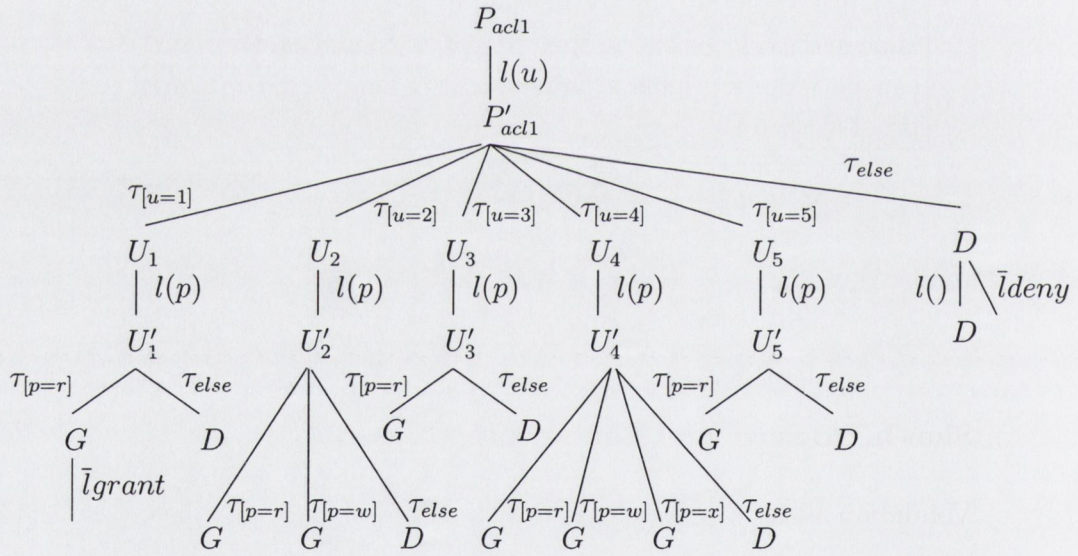


Fig. 7.5: Tree of Policy Version 1

7.4.2 Proving Access Control Model Flexibility

The previous section demonstrated that the access control models can be expressed using the π calculus. Hereby the access control model process expression P was understood to be embedded into the object server, with the client accessing the object

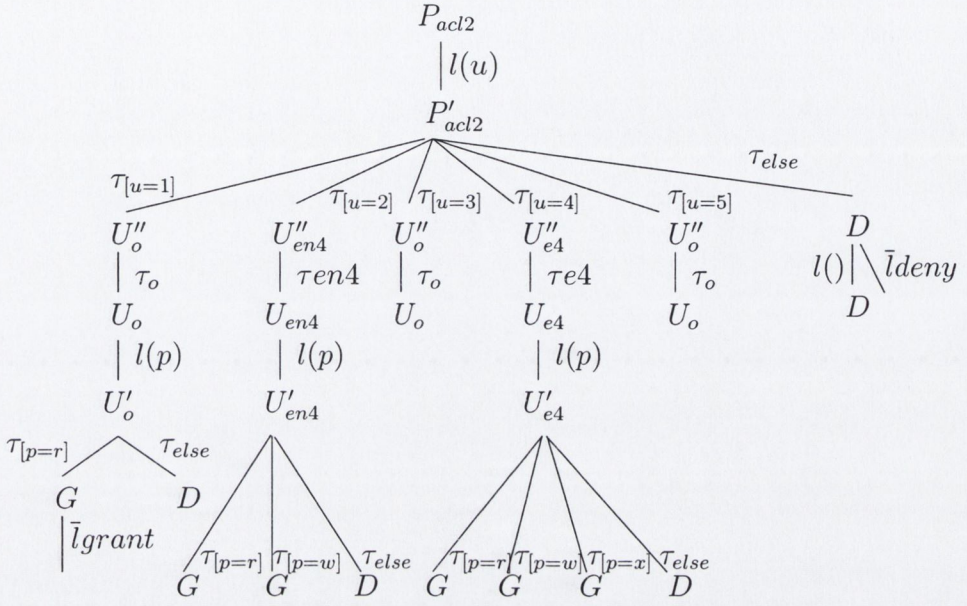


Fig. 7.6: Tree of Policy Version 2

server directly. This setup can be seen as the traditional, correct instantiation of the access control model.

Figure 7.7 shows an abstract view of three different access control mechanisms. C_1 represents the client on the left side, and the server is situated on the right. A_{scap} is the access control mechanism proxy in the middle area of the figure.

Figure 7.7–(1) shows the hybrid setup, which can be seen as the basic form. The full policy and authentication implementation is situated in the server. The client accesses the server directly. Security in this system can be shown by verifying the server implementation. This can be identified as the current practice, which requires a customised implementation of the access control mechanism in each server.

Figure 7.7–(2) shows the proxy based setup. In a proxy based system the server and client are separated by the ASCap proxy. The server does not have the policy, but gets the policy object from the ASCap proxy. It has been noted that using the abstraction of communication channels, which are becoming established, is equivalent

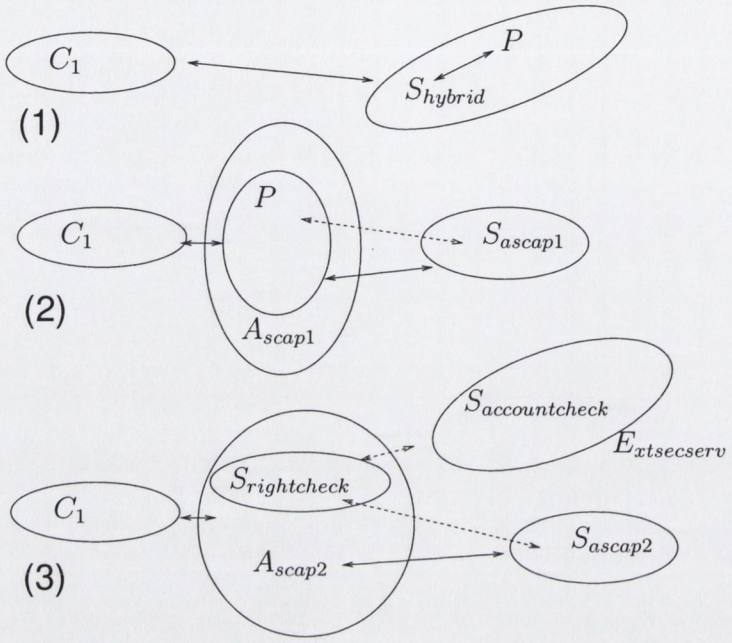


Fig. 7.7: The Different Access Control Mechanism.

to the notion of sending mobile code from the scope of one process to another [100]. Therefore we will model the transmission of the policy object as creation of a new channel between the policy and the object server. In this setup security depends not only on the implementation, but also interaction of the different framework elements. If it can be shown, that the proxy based setup behaves equivalently to the hybrid setup we have proven its flexibility and security.

Figure 7.7–(3) shows the external server based setup, which is achieved by extending the proxy based setup with external security servers. In this setup the policy will be split into several parts, but using the proof of Section 7.4.1 it can be shown that $(S_{rightcheck}|S_{accountcheck} = P_{acl2}) \approx P_{acl}$. This is a substantial result as it shows that the π-calculus can be used to show behavioural equivalents of setups with an unequal number of policy parts.

In the following section we will derive process expressions for each of these systems and we will see the hybrid system as a measurement for the correct system behaviour. Therefore, our proof will show behavioural equivalence of the two extended versions to the simple hybrid version.

Deriving the π-Calculus Expressions

In the following we will derive process expressions for a hybrid (seen to be conventional system) the client process expression and the two unified access control mechanism.

Hybrid System

$$S_{hybrid} \stackrel{def}{=} (\nu l)(\overline{c}l|P)$$

The server has the policy implementation P (e.g., P_{acl1}), as well as the l channel to access it in its private scope. Upon an access request the server extends the scope of the l channel to the client. The policy may have a format like that described in Section 7.4.1 taking the user-id and requested right as input and sending a grant or deny on the same channel as output. It is also conceivable that a system expression describes the full authentication protocol including use of cryptography, as suggested by Abadi et al. [1]. However this will be neglected for our behaviour equivalence proof.

Client Process

$$C_1 \stackrel{def}{=} c(n).\overline{n}1.\overline{n}r.n(x).[x = grant]\overline{passed}$$

The client process expression is like the test expression in Section 7.4.1. The difference is that first a new channel name is received, which will be used as login channel.

Proxy Based Setup

This is the first of two unified framework setups.

ASCap Proxy in Proxy Based

The proxy based setup consists only of a server and an ASCap proxy. The ASCap proxy consists of three functional parts.

- Secure Server Connection Setup
- Policy-Server Interface
- Client-Server Interface

A secure connection is setup to an available server $(\nu m)(\overline{c_{sa}}m.\overline{m}c_{sp}.\overline{m}c_{sl})$. The channel c_{sa} is used to provide the two communication channels (sl=Server login-information; sp=Server Policy). The restriction of m can be implemented for example by a fresh symmetric key. The policy implementation residing in the ASCap proxy needs to be accessible by the server $((\nu l)(\overline{c_{sp}}(l)|P))$. In practice scope extrusion can be implemented in different ways, either the server receives a reference to the policy implementation, which is residing on a secure repository, or a signed full policy object (mobile code) can be transferred. This aspect is also described by Milner in [100].

Finally, the login information from the client interface needs to be made accessible to the server $(c_{sl}(b).\bar{c}b)$. The full ASCap proxy process expression reads as follows:

$$A_{scap} \stackrel{def}{=} (\nu c_{sp}, c_{sl})\,[(\nu m)(\overline{c_{sa}}m.\overline{m}c_{sp}.\overline{m}c_{sl}) \quad | \quad (\nu l)(\overline{c_{sp}}l|P) \quad | \quad (c_{sl}(b).\bar{c}b)]$$

Server in Proxy Based Setup

In the proxy based setup the server process expression needs to adapt to the policy and client connection being provided by the ASCap proxy.

$$S_{ascap1} \stackrel{def}{=} c_{sa}(d).d(p).d(q).p(a).\bar{q}a$$

The sever receives the fresh channels on c_{sa} and via the new channel p and q. Then in the simplest case the server provides the client (q) with the login channel received from the policy (p). This means a new channel between the client and policy is created by the server.

External Security Server Based

This is the second unified framework. It is an extension of the proxy based setup using external security servers to further outsource part of the policy behaviour.

ASCap Proxy in External Security Server Based

The ASCap Proxy in the external security server based setup is extended with the external security server communication part. In an implementation this would require additional connection setup code and signature checking. The core process is

$$(\nu h)(c_{ess}(a).c_{ess}(a').c_{sl}(d).c_{sl}(d').\overline{c}h.h(b).\overline{a'}b.a(x).\overline{d}x.h(b').\overline{d'}b'),$$

which provides a login information channel to the client, $\overline{c}h$, receives information on it, $h(b)$, and redirects it ($c_{ess}(a).h(b)$ and $\overline{a'}b$) to either the $S_{accountcheck}$ part of the policy (resided in the external security server), or $S_{rightcheck}$ ($c_{sl}(d')$ and $h(b').\overline{d'}b'$). Further more, the two policy parts are required to interact on a private channel k, which is handled by $c_{ess}(a').c_{sl}(d)$ and $a(x).\overline{d}x$. It becomes obvious, that once a policy has a different interaction pattern (e.g., requires more secret channels, or different login information at different points), this part of the ASCap proxy has to be adapted. Therefore in the following, the proof of equivalent behaviour can only be given for the class of policies of the form $S_{rightcheck}$ of form $l'(x).\cdots.\overline{k'}$ and $S_{rightcheck}$ of form $k(y).\cdots.l(z)$. The full ASCap proxy process expression is given below:

$$A_{scap2} \stackrel{def}{=} (\nu c_{sl}, c_{sp})((\nu h)(c_{ess}(a).c_{ess}(a').c_{sl}(d).c_{sl}(d').\bar{c}h.h(b).$$
$$\overline{a'}b.a(x).\bar{d}x.h(b').\overline{d'}b')_h|$$
$$(\nu l, k)(\overline{c_{sp}}k.\overline{c_{sp}}l|S_{rightcheck})|\overline{c_{sa}}c_{sp}.\overline{c_{sa}}c_{sl})_{c_{sl},c_{sp}}$$

External Security Server in External Security Server Based

An external security server has a form similar to the policy interface part of an ASCap proxy.

$$E_{xtsecserv} \stackrel{def}{=} (\nu l, k)(\overline{c_{ess}}k.\overline{c_{ess}}l|S_{accountcheck})$$

Server in External Security Server Based

The particularity, that the policy is split into two parts is also reflected in the server process expression.

$$S_{ascap2} \stackrel{def}{=} c_{sa}(p).c_{sa}(q).p(a).\bar{q}a.p(a).\bar{q}a$$

Generally the server will be required to handle as many channels as policy parts exist in the system.

Showing Behavioural Equivalence Of Hybrid and Version 1

In this section the process expressions will be rewritten using the axiomatisation system of Section A.1 to show weak late congruence. First, the process expressions are copied for ease of reading:

$$A_{scap} \stackrel{def}{=} (\nu c_{sp}, c_{sl})((\nu m)((\overline{c_{sa}}(m).\overline{m}c_{sp}.\overline{m}c_{sl}) \mid (\nu l)(\overline{c_{sp}}(l)|P_{acl1}) \mid (c_{sl}(b).\bar{c}b)$$

$$S_{ascap1} \stackrel{def}{=} c_{sa}(d).d(p).d(q).p(a).\bar{q}a$$

Then we would like to introduce an axiom to put a policy P out of a scope of a restriction:

$$SA \quad (\nu l)(\overline{c_{sp}}(l)|P) \approx_l^c (\nu l)(\overline{c_{sp}}(l).P) \; if \; P \; of \; form \; l(x).\cdots$$

To prove this axiom it has to be noted, that P cannot do a transition before $\overline{c_{sp}}(l)$ extends the scope of l, thus allowing other expressions to communicate with P.
In the following, the commented sequence of equations are given:

$$(\nu c_{sa})(A_{scap} \quad | \quad S_{ascap1})$$

We restrict c_{sa} as we only consider the case in which this A_{scap} and S_{ascap1} use c_{sa} to communicate with each other instead of with other processes. This restriction can be implemented using public key cryptography. Note we are using the abbreviated form for the restriction (νm).

$$(\nu c_{sa})((\nu c_{sp}, c_{sl}) \quad (\quad \overbrace{(\overline{c_{sa}}(m).\overline{m}(c_{sp}).\overline{m}(c_{sl}))}^{A} | \overbrace{(\nu l)(\overline{c_{sp}}(l)|P_{acl1}) \quad | \quad (c_{sl}(b).\overline{c}b))}^{B} | \overbrace{c_{sa}(d).d(p).d(q).p(a).\overline{q}a)}^{C}$$

It has to be kept in mind that every $(\nu c_{sp}, c_{sl})$ in the expansion is the same as in the wider scope. Using E on $(\nu c_{sp}, c_{sl})(B|\overbrace{A}^{1})|C$ with $P = \overline{c_{sa}}(m).\overline{m}(c_{sp}).\overline{m}(c_{sl})$ and $P' = c_{sa}(d).d(p).d(q).p(a).\overline{q}a.$

$$
\begin{aligned}
1)P|P' \;=\; & (\overline{c_{sa}}(m).(\overline{m}(c_{sp}).\overline{m}(c_{sl})|c_{sa}(d).d(p).d(q).p(a).\bar{q}a) + \\
& c_{sa}(d).(\overline{c_{sa}}(m).\overline{m}(c_{sp}).\overline{m}(c_{sl})|d(p).d(q).p(a).\bar{q}a) + \\
& ([c_{sa}=c_{sa}])\tau(\nu m)(\overbrace{\overline{m}(c_{sp}).\overline{m}(c_{sl})|m(p).m(q).p(a).\bar{q}a}^{2}))
\end{aligned}
$$

After this expansion it can be seen that the system can proceed in three ways. Either the proxy establishes a connection with an arbitrary server (by doing $\overline{c_{sa}}(m)$); Or the server receives a connection from a different proxy ($c_{sa}(a)$); or both interact with each other. As we restricted c_{sa} we can rule out the first two cases by using RES4,RES2 and RES1.
Now we apply E to 2 with $P = \overline{m}(c_{sp}).\overline{m}(c_{sl})$
and $P' = m(p).m(q).p(a).\bar{q}a$.

$$
\begin{aligned}
2)P|P' \;=\; & \overline{m}(c_{sp}).(\overline{m}(c_{sl})|m(p).m(q).p(a).\bar{q}a) + \\
& m(p).(m(q).p(a).\bar{q}a|\overline{m}(c_{sp}).\overline{m}(c_{sl})) + \\
& \tau.(\nu c_{sp})\overbrace{(\overline{m}(c_{sl})|m(q).c_{sp}(a).\bar{q}a)}^{3}
\end{aligned}
$$

Although the expansion results in three nondeterministic choices, the upper two can be removed by RES4 and RES2 thus we expand (3) further.
The (νc_{sp}) is the same as in wider scope.
E with $P = \overline{m}(c_{sl})$ and $P' = m(q).c_{sp}(a).\bar{q}a$.

$$
\begin{aligned}
3)P|P' \;=\; & \overline{m}(c_{sl}).(0|m(q).c_{sp}(a).\bar{q}a) + \\
& m(q).(c_{sp}(a).\bar{q}a|\overline{m}(c_{sl})) + \\
& \tau.(\nu c_{sl})(0|c_{sp}(a).\overline{c_{sl}}a)
\end{aligned}
$$

The upper two choices can be removed using RES4, RES2 and RES1. We need to recall, that E in 1) used the form $(\nu c_{sp}, c_{sl})(B|A)|C$ with $A|C$ like $(1(2(3)))$. We rewrite the original term now as:

$$(\nu c_{sp}, c_{sl}) \quad (\quad (\nu l)(\overline{c_{sp}}l|P_{acl1}) \quad | \quad (c_{sl}(b).\overline{c}b) \quad |$$
$$\tau.(\nu m)\tau.(\nu c_{sp})\tau.(\nu c_{sl})(0|c_{sp}(a).\overline{c_{sl}}a))$$

We remember that the scope of the restrictions $(\nu c_{sp}, c_{sl})$are the same as in the wider scope, hence we don not need to write it twice . We apply RES3 and RES4 to remove (νm). TAU1 can be used to remove the τ.

$$(\nu c_{sp}, c_{sl}) \quad (\quad (\nu l)(\overline{c_{sp}}(l)|P_{acl1})|(c_{sl}(b).\overline{c}b) \quad | \quad c_{sp}(a).\overline{c_{sl}}a)$$

Before we can apply the E rule we need to put P_{acl1} out of the scope of (νl). This can be done by applying SA of above. E can now be used with $P = \overline{c_{sp}}(l).P_{acl1}$ and $P' = c_{sp}(a).\overline{c_{sl}}a$.

$$\begin{aligned} 4)P|P' \quad = \quad & (\overline{c_{sp}}(l).(P_{acl1}|c_{sp}(a).\overline{c_{sl}}a + \\ & c_{sp}(a).(\overline{c_{sp}}(l).P_{acl1}|\overline{c_{sl}}a + \\ & \tau.(\nu l)(P_{acl1}|\overline{c_{sl}}(l))) \end{aligned}$$

Upper two are 0 according to RES4,RES2 and RES1.
After applying TAU1 and SA use E rule with $P = \overline{c_{sl}}(l).P_{acl1}$ and $P' = c_{sl}(b).\overline{c}b$.

$$\begin{aligned} 5)P|P' \quad = \quad & \overline{c_{sl}}(l).(P_{acl1}|c_{sl}(b).\overline{c}b) + \\ & c_{sl}(b).(\overline{c_{sl}}(l).P_{acl1}|\overline{c}b) + \\ & \tau.(\nu l)(P_{acl1}|\overline{c}(l))) \end{aligned}$$

Upper two are 0 according to RES4,RES2 and RES1. τ removed by TAU1. Then use (4) and (5) to rewrite the original term and remove $(\nu c_{sp}, c_{sl})$ by RES3 and RES4. We get:

$$(\nu l) \qquad \overline{c}(l)|P_{acl1}$$

Recalling that $S_{hybrid} \stackrel{def}{=} (\nu l)(\overline{c}l|P_{acl1})$ we can now see that

$$(\nu l)\overline{c}(l)|P_{acl1} \approx_l^c (\nu l)(\overline{c}l|P_{acl1})$$

means

$$A_{scap}|S_{ascap1} \approx_l^c S_{hybrid}.$$

This result can be lifted to

$$!A_{scap}|!S_{ascap1} \approx_l^c !S_{hybrid}$$

by showing that the initial communication on c_{sa} between A_{scap} and S_{ascap1} extends the scope of the fresh m (and later c_{sp}, c_{sl}). m, c_{sp}, c_{sl} being secret, prevents instances of A_{scap} and S_{ascap1}, which are in the same stage of communication to interfere with each other.

Informally this means that infinite parallel running instances of the A_{scap} proxy and infinite simultaneous running instances of the S_{ascap1} are weak late congruent with an infinite number of S_{hybrid} waiting for communication, provided both use the same policy P as the access decision function.

This result holds because in the equational reformulation we used only rules, which do not modify weak late congruence. However we have to keep in mind that after using SA, our system is restricted to policies that start with an input action on channel l before freely interacting with the environment. Clearly, this limitation is of no burden to the access control models, which can be expressed.

Showing Behavioural Equivalence Of Hybrid and Version 2

This section shows behavioural equivalence of the external security server based system described by the process expressions given before:

$$\begin{aligned} A_{scap2} \stackrel{def}{=} \; & (\nu c_{sl}, c_{sp})((\nu h)(c_{ess}(a).c_{ess}(a').c_{sl}(d).c_{sl}(d').\overline{c}(h).h(b).\overline{a'}b.a(x).\overline{d}x. \\ & h(b').\overline{d'}b')_h|(\nu l, k)(\overline{c_{sp}}(k).\overline{c_{sp}}(l)|S_{rightcheck}) \quad | \\ & \overline{c_{sa}}(c_{sp}).\overline{c_{sa}}(c_{sl}))_{c_{sl},c_{sp}} \end{aligned}$$

$$E_{xtsecserv} \stackrel{def}{=} (\nu l, k)(\overline{c_{ess}}(k).\overline{c_{ess}}(l)|S_{accountcheck})$$

$$S_{ascap2} \stackrel{def}{=} c_{sa}(p).c_{sa}(q).p(a).\overline{q}a.p(a).\overline{q}a$$

In the equations below the braces are augmented with indices noting the scope of restrictions ending by this brace. Also partial expressions are omitted, if it can be

shown that they can be reduced to the empty process. Section 7.4.2 can be used to see examples of the full terms.

$$
\begin{aligned}
&(\nu c_{sa})((\nu c_{ess})(A_{scap2}|E_{xtsecserv}) \quad | \quad S_{ascap2}) \\
&(\nu c_{sa})((\nu c_{ess})((\nu c_{sl}, c_{sp})(\quad | \quad c_{sa}(p).c_{sa}(q).p(a).\overline{q}a.p(a).\overline{q}a)_{c_{sa}} \\
&(\nu h)(c_{ess}(a).c_{ess}(a').c_{sl}(d).c_{sl}(d').\overline{c}(h).h(b). \\
&\qquad \overline{a'}b.a(x).\overline{d}x.h(b').\overline{d'}b')_h| \\
&\qquad (\nu l, k)(\overline{c_{sp}}(k).\overline{c_{sp}}(l)|S_{rightcheck})| \\
&\qquad \overline{c_{sa}}(c_{sp}).\overline{c_{sa}}(c_{sl}))_{c_{sl},c_{sp}} \\
&\qquad (\nu l, k)(\overline{c_{ess}}(k).\overline{c_{ess}}(l)|S_{accountcheck}))_{c_{ess}}
\end{aligned}
$$

Renaming l & k .

$$
\begin{aligned}
&(\nu c_{sa})((\nu c_{ess})((\nu c_{sl}, c_{sp})(\quad | \quad c_{sa}(p).c_{sa}(q).p(a).\overline{q}a.p(a).\overline{q}a)_{c_{sa}} \\
&(\nu h)(c_{ess}(a).c_{ess}(a').c_{sl}(d).c_{sl}(d').\overline{c}(h).h(b). \\
&\qquad \overline{a'}b.a(x).\overline{d}x.h(b').\overline{d'}b')_h| \\
&\qquad (\nu l, k)(\overline{c_{sp}}(k).\overline{c_{sp}}(l)|S_{rightcheck})| \\
&\qquad \overline{c_{sa}}(c_{sp}).\overline{c_{sa}}(c_{sl}))_{c_{sl},c_{sp}} \\
&\qquad (\nu l', k')(\overline{c_{ess}}(k').\overline{c_{ess}}(l')|S_{accountcheck}))_{c_{ess}}
\end{aligned}
$$

E with $P' = \overline{c_{ess}}(k').\overline{c_{ess}}(l')$ and $P = c_{ess}(a).c_{ess}(a').c_{sl}(d).c_{sl}(d').\overline{c}(h).h(b).\overline{a'}b.a(x).\overline{d}x.h(b').\overline{d'}b'$.)

$$
\begin{aligned}
P|P' = (&c_{ess}(a).\cdots+\overline{c_{ess}}(k').\cdots+ \\
&\tau. \\
&(\nu k')c_{ess}(a').c_{sl}(d).c_{sl}(d'). \\
&\overline{c}(h).h(b).\overline{a'}b.k'(x).\overline{d}x.h(b').\overline{d'}b'|\overline{c_{ess}}(l'))
\end{aligned}
$$

The first two choices can be reduced to 0 by RES4, RES2 and RES1. E with $P = c_{ess}(a').c_{sl}(d).c_{sl}(d').\overline{c}(h).h(b).\overline{a'}b.k'(x).\overline{d}x.h(b').\overline{d'}b'$ and $P_j = \overline{c_{ess}}(l').0$.

$$P|P' = (c_{ess}(a').\cdots+\overline{c_{ess}}(l').\cdots+$$
$$\tau.$$
$$(\nu l')c_{sl}(d).c_{sl}(d').$$
$$\overline{c}(h).h(b).\overline{l'}b.k'(x).\overline{d}x.h(b').\overline{d'}b'|0)$$

The first two choices can be reduced to 0 by RES4, RES2 and RES1.
Hence, the whole expression after further simplifications becomes:

$$(\nu c_{sa})((\nu c_{ess})((\nu c_{sl}, c_{sp})((\nu h)($$
$$(\nu l', k')(\tau.\tau.c_{sl}(d).c_{sl}(d'). \quad | \quad c_{sa}(p).c_{sa}(q).p(a).\overline{q}a.p(a).\overline{q}a)_{c_{sa}}$$
$$\overline{c}(h).h(b).\overline{l'}b.k'(x).\overline{d}x.h(b').\overline{d'}b')_h|$$
$$S_{accountcheck})_{l',k'}|$$
$$(\nu l, k)(\overline{c_{sp}}(k).\overline{c_{sp}}(l)|S_{rightcheck})|$$
$$\overline{c_{sa}}(c_{sp}).\overline{c_{sa}}(c_{sl}))_{c_{sl},c_{sp}}$$

E with $P = \overline{c_{sa}}(c_{sp}).\overline{c_{sa}}(c_{sl})$ and $P' = c_{sa}(p).c_{sa}(q).p(a).\overline{q}a.p(a).\overline{q}a.$

$$P|P' = \overline{c_{sa}}(c_{sp}).\cdots+c_{sa}(p).\cdots+$$
$$\tau.(\nu c_{sp})\overline{c_{sa}}(c_{sl})|$$
$$c_{sa}(q).c_{sp}(a).\overline{q}a.c_{sp}(a).\overline{q}a)$$

The first two choices can be reduced to 0 by RES4, RES2 and RES1.
E with $P = \overline{c_{sa}}(c_{sl})$ and $P' = c_{sa}(q).c_{sp}(a).\overline{q}a.c_{sp}(a).\overline{q}a.$

$$P|P' = \overline{c_{sa}}(c_{sl}).\cdots+c_{sa}(q).\cdots+$$
$$\tau.(\nu c_{sl})0|$$
$$c_{sp}(a).\overline{c_{sl}}a.c_{sp}(a).\overline{c_{sl}}a)$$

The first two choices can be reduced to 0 by RES4, RES2 and RES1.
After further simplifications the full expression becomes.

$$(\nu c_{sa})((\nu c_{ess})((\nu c_{sl}, c_{sp})((\nu h)($$
$$(\nu l', k')(\tau.\tau.c_{sl}(d).c_{sl}(d'). \quad | \quad \tau.\tau.c_{sp}(a).\overline{c_{sl}}a.c_{sp}(a).\overline{c_{sl}}a)_{c_{sl},c_{sp}})_{c_{sa}}$$
$$\overline{c}(h).h(b).\overline{l'}b.k'(x).\overline{d}x.h(b').\overline{d'}b')_h|$$
$$S_{accountcheck})_{l',k'}|$$
$$(\nu l, k)(\overline{c_{sp}}(k).\overline{c_{sp}}(l)|S_{rightcheck})$$

E with $P = \overline{c_{sp}}(k).\overline{c_{sp}}(l)$ and $P' = c_{sp}(a).\overline{c_{sl}}a.c_{sp}(a).\overline{c_{sl}}a.$

$$P|P' = \overline{c_{sp}}(k).\cdots + c_{sp}(a).\cdots +$$
$$\tau.$$
$$(\nu k)(\overline{c_{sp}}(l)|\overline{c_{sl}}(k).c_{sp}(k).\overline{c_{sl}}(k)$$

The first two choices can be removed using RES4, RES2 and RES1.
Hence the full term reads:

$$(\nu c_{sa})((\nu c_{ess})((\nu c_{sl}, c_{sp}, k)((\nu h)($$
$$(\nu l', k')(\tau.\tau.c_{sl}(d).c_{sl}(d'). \quad | \quad \tau.\tau.\tau\overline{c_{sl}}(k).c_{sp}(k).\overline{c_{sl}}(k))_{c_{sl},c_{sp},k})_{c_{sa}}$$
$$\overline{c}(h).h(b).\overline{l'}b.k'(x).\overline{d}x.h(b').\overline{d'}b')_h|$$
$$S_{accountcheck})_{l',k'}|$$
$$(\nu l)(\overline{c_{sp}}(l)|S_{rightcheck})$$

E with $P = c_{sl}(d).c_{sl}(d').\overline{c}(h).h(b).\overline{l'}b.k'(x).\overline{d}x.h(b').\overline{d'}b'$ and $P' = \overline{c_{sl}}(k).c_{sp}(k).\overline{c_{sl}}(k)$.

$$P|P' = c_{sl}(d).\cdots + \overline{c_{sl}}(k).\cdots +$$
$$\tau.(\nu k)c_{sl}(d').\overline{c}(h).$$
$$h(b).\overline{l'}b.k'(x).\overline{k}x.h(b').\overline{d'}b'|$$
$$c_{sp}(k).\overline{c_{sl}}(k))$$

The first two choices can be removed using RES4, RES2 and RES1.
Hence the full term reads:

$$(\nu c_{sa})((\nu c_{ess})((\nu c_{sl}, c_{sp}, k)((\nu h)($$
$$(\nu l', k')(\tau.\tau.\tau.c_{sl}(d'). \quad | \quad \tau.\tau.\tau c_{sp}(k).\overline{c_{sl}}(k))_{c_{sl},c_{sp},k})_{c_{sa}}$$
$$\overline{c}(h).h(b).\overline{l'}b.k'(x).\overline{k}x.h(b').\overline{d'}b')_h|$$
$$S_{accountcheck})_{l',k'}|$$
$$(\nu l)(\overline{c_{sp}}(l)|S_{rightcheck})$$

E with $P = c_{sp}(k).\overline{c_{sl}}(k)$ and $P' = \overline{c_{sp}}(l)$.

$$P|P' = c_{sp}(k).\cdots + \overline{c_{sp}}(l).\cdots +$$
$$\tau.(\nu l)\overline{c_{sl}}(l)|0)$$

The first two choices can be removed using RES4, RES2 and RES1.
Hence the full term reads:

$$(\nu c_{sa})((\nu c_{ess})((\nu c_{sl}, c_{sp}, k, l)((\nu h)($$
$$(\nu l', k')(\tau.\tau.\tau.c_{sl}(d'). \quad | \quad \tau.\tau.\tau.\tau\overline{c_{sl}}(l))_{c_{sl},c_{sp},k,l})_{c_{sa}}$$
$$\overline{c}(h).h(b).\overline{l'}b.k'(x).\overline{k}x.h(b').\overline{d'}b')_h|$$
$$S_{accountcheck})_{l',k'}|$$
$$S_{rightcheck})$$

E with $P = c_{sl}(d').\overline{c}(h).h(b).\overline{l'}b.k'(x).\overline{k}x.h(b').\overline{d'}b'$ and $P' = \overline{c_{sl}}(l).$

$$P|P' = c_{sl}(d'). \cdots + \overline{c_{sl}}(l). \cdots +$$
$$\tau.(\nu l)\overline{c}(h).h(b).$$
$$\overline{l'}b.k'(x).\overline{k}x.h(b').\overline{l}b'|0)$$

The first two choices can be removed using RES4, RES2 and RES1.
Hence the full term reads:

$$(\nu c_{sa})((\nu c_{ess})((\nu c_{sl}, c_{sp}, k, l)((\nu h)($$
$$(\nu l', k')(\tau.\tau.\tau.\tau. \quad | \quad \tau.\tau.\tau.\tau.0)_{c_{sl},c_{sp},k,l})_{c_{sa}}$$
$$\overline{c}(h).h(b).\overline{l'}b.k'(x).\overline{k}x.h(b').\overline{l}b')_h|$$
$$S_{accountcheck})_{l',k'}| \quad S_{rightcheck})$$

Using RES1-RES4 some restrictions can be removed. TAU1 removes the τ.

$$(\nu k, l)((\nu h)((\nu l', k')(\overline{c}(h).$$
$$h(b).\overline{l'}b.k'(x).\overline{k}x.h(b').\overline{l}b')_h \quad | \quad S_{accountcheck})_{l',k'}|S_{rightcheck})_{k,l}$$

$$(\nu k, l)((\nu h)((\nu l', k') \quad (\quad \overbrace{\overline{c}(h).h(b).\overline{l'}b}^{A}.\overbrace{k'(x).\overline{k}x}^{B}.\overbrace{h(b').\overline{l}b'}^{C})_h|$$
$$S_{accountcheck})_{l',k'}|S_{rightcheck})_{k,l}$$

Notable in this process expression is that $S_{accountcheck}$ and $S_{rightcheck}$ have no direct communication channel (i.e., they reside in different servers). From Section 7.4.1 we know that $S_{accountcheck}$ requires communication on l' before it does internal communi-

cation on k'. The part A receives the login information from the client and sends it to $S_{accountcheck}$. B transfers the internal information from $S_{accountcheck}$ to $S_{rightcheck}$ and C provides $S_{rightcheck}$ with the next information received from the client.
If $S_{accountcheck}$ has the form $l'(x).\cdots.\overline{k'}$ and $S_{rightcheck}$ has the form $k(y).\cdots.l(z)$ then the system can be shown to be $\approx_l^c S_{hybrid}$ using the bisimulation given in Section 7.4.1.

7.4.3 Discussion of the Formal Evaluation

This section presented a proof for the flexibility of the ASCap framework using the π-calculus. After showing how to express the access matrix model using the mismatch operator, a bisimulation was used to show that two different model specifications are behaviouraly equivalent. This highlights the π-calculus suitability to express access control models, as well as the fact it is possible to reason about behavioural equivalence. Then the process expressions of the ASCap framework were derived. For the proof we also define a hybrid setup, in which the full access control model implementation is hardwired into the object server, which represents today's customised access control mechanims. Then two versions of the ASCap framework were presented. The first version (proxy based) introduced dynamic policy changes by migrating the location where the policy is stored. The second version extended the setup by external security servers, which allows the policy to be split into parts, to be stored and administrated by different entities. Using equational reasoning we showed weak late congruents of each version of the ASCap framework to the hybrid setup, which can be understood as the ASCap framework not altering the system behaviour while introducing additional flexibility. However, the proof puts the constraint onto the access control model expression to start with a read on the l channel, such as reading in the client id first. We like to note that this constraint does not affect the ability of the access control models supported, because virtually all models require at least one input parameter to the policy, which just needs to be transfered before other communication.

7.5 Evaluation of Integrating the ASCap Framework into Intrusion Detection Systems[5]

Once the paradigm shift to "partial outsourcing of access control is possible" (cf. Section 5.8.5) is fully realised, new benefits can be seen. This section highlights one of these benefits by presenting novel intrusion detection strategies.

Currently intrusion detection systems (IDS) monitor network traffic and either compare the traffic with alarm signatures in their database or employ neural networks to do abnormality detection. Both techniques have drawbacks on both the false positive or false negative side: either unknown attack signatures pass undetected or the volume of alarms is too high to be useful.

Newer IDS systems are connected with the local firewall and form active intrusion prevention systems (IPS). IPS are called *active* because they are able to actively modify network traffic, rendering well-known exploits and backdoor techniques obsolete. However, we claim the technique to detect the intrusions still employs passive recognition via signature databases.

We argue that partial outsourcing allows us to connect the access control framework with the IDS and thus *active recognition* strategies become possible. We will present in three different examples, the benefits of using partial outsourcing.

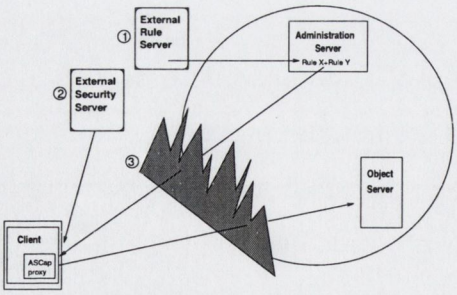


Fig. 7.8: An Example Scenario

[5] **A version of this section was published in an earlier paper [4].**

The scenario used in our evaluation is shown in Figure 7.8. It includes the client which requests access to objects managed by the object server and an internal administration server that defines the security policies enforced by the ASCap framework. The scenario also includes an external rule server (1), a possible external security server (2), and a firewall (3).

Remote Weakness Filtering

In this case, the external consulting company actively follows security announcements and weakness reports. Whenever an exploit is reported, auditing of customer companies is done and security holes are recorded. However, the external company does not have the power to update the actual software, nor the local administration, during office hours to guarantee a timely reaction on their notification. To prevent intrusion during this blackout time, one would want to switch off the vulnerable applications. This can be done in two ways: either the external consulting company employs an external security server, which stops handing out valid credentials (class γ of Section 5.8.5), or the external consulting company updates rules (class δ) for the respective applications to strictly deny access. The latter approach would have the advantage that, instead of a full blackout, only those access requests possibly touching the security hole will be blocked. This is similar to only filtering emails that carry a virus. In an ideal case, security holes could be closed by patching the ASCap framework preventing dangerous access requests from getting through.

We simulated this example by having an external rule server transfer rule objects to the administration server's rule-incoming directory via *scp*[6]. The JINI lease time was set to 30 seconds, which might seem short, but because the JINI look-up server was running on the administration server it turned out to be feasible and gave us an acceptable revocation time.

[6] A network copy tool of the public ssh implementation [172].

A remote exploitable weakness was assumed to be a read request with an information location string longer than 100 bytes. As expected, it was possible to introduce an appropriate check into the system by uploading its implementation as a rule object to the administration server.

Advanced Pattern Recognition

Advanced patterns can be the usual behavioural pattern of a client but under different circumstances such as different access control environments. An ideal access control environment would always want to employ the most secure countermeasures, such as multiple independent authentication protocols and repeated credential requests to prevent unobserved terminals being misused (e.g. during the lunch break). However always using the most severe countermeasures could harm productivity and alienate the user, ultimately resulting in less security as security measures will be deliberately disabled (e.g. using an automatic Password Completion Tool). A solution could be to create different equally secure scenarios and employ them alternately. In one scenario the authentication is done by password, while in another, maybe during lunchtime, it would require an SMS reply. One benefit is that if one of these countermeasures breaks (e.g. sharing of a password) the system will not necessarily fail. However, the real benefit will be to record behavioural pattern of each scenario separately. It is safe to assume the same user will exhibit a similar pattern in all scenarios, therefore, if the pattern varies, an alarm can be raised and a closer investigation started.

Another case of advanced patterns are not so easily seen from a local perspective. An example is SPAM filtering of email systems. Locally, a domain would not be able to recognise certain spammers, because only a few mails are delivered to this local domain. Being able to monitor email traffic from a broader perspective it is possible to recognise these SPAM originators. Applying this technique to access control translates to monitoring the access request per client over a given period of time. Doing this not on

the local, but on a broader scale allows the recognition of clients randomly 'hacking' into different systems, (e.g. by using a newly discovered exploit and collecting an extraordinary number of access rights).

We implemented a prototype that employed the external security servers of the AS-Cap framework, thus instantiating class γ (cf. Section 5.8.5). In addition to the standard rules, two different rules, which access external monitoring services were plugged into the system. This was done by altering the policy object to include a rule, which requests the credential of this external security server. We implemented a counter on the external security server, which would deny the credential if too many different information directories were accessed in a short amount of time. We assumed that this corresponded to different security domains in a real world setup. Further we assumed that one external monitoring service was fully trusted, while the second should only be allowed notification character. Hence, although the behaviour of both external security servers were the same, the rule implementation of checking their required credentials was different. The fully trusted server was able to cause a denial of service (also if it would only be unreachability) while the second's server credentials were not further checked or required, thus it could not cause a denial of service. The added value was that this external security server would be able to collect statistical information about clients accessing our servers.

Active Intruder Recognition

Reasoning about intruders shows that different properties can be recognised [160]. An external company can specialise in providing tests, which unveil potential intruders and industrial spies. Partial outsourcing allows a company to benefit from external tests, which not only consist of passive recognition but active probing without loosing sovereignty.

Assuming an intruder has, as opposed to a 'normal' user, a higher desire to get a

service, maybe even any service, his behaviour will differ from the normal user. An appropriate test would be to temporarily close down part of the system for this user, if the user keeps on downloading files from a still open part, an industrial spy might be found.

Or, in a different case, the intruder may be restricted to security hole exploiting techniques to acquire access rights.

These assumptions can be used to actively recognise intruders without exactly knowing which security hole they used. Given that the system finds itself in an alarmed state (e.g. an unnaturally high system load) it may start to change the environment in a subtle way. The changes will not disturb official users, but cause trouble to intruders. A firewall (Figure 7.8–3) might disallow a certain protocol or require a reinitiating of the connection. Connections established by a replay attack will become obvious or hijacked credentials will have timed out. Generally, if after such a change in access control rules a user shows different behaviour than before or even does not connect again, a new security hole might have been found. Active intrusion recognition is not automated to this extent today, but by allowing partial outsourcing it becomes possible to develop such a mechanism.

Note: This example has not been implemented, but all required functionalities exist in the ASCap framework. To fully implement this example, research about active intruder recognition techniques would need to be conducted. Research into intrusion detection is mostly restricted to passive pattern recognition instead of active probing.

Comparison of Example Implementations

Below in Table 7.3 we have compared the examples, their outsourcing class and employed elements of the ASCap framework. The comments review the key elements of each case.

Example	Class	Ext. Security Server	Ext. Rule Server	Tested	Comment
4.1 Remote Weakness Filtering	γ	x	-	-	External security server stops handing out credentials to block access
4.1 Remote Weakness Filtering	δ	-	x	x	External rule server updates rule implementation on administration server, which updates policy objects
4.2 Advanced Pattern Recognition	γ	x	-	x	External security server receives statistical information as clients need to contact for access. If pattern is recognised credential can be denied and may cause a blockout of the client depending on the actual rule.
4.3 Active Intruder Recognition	δ	x	x	-	Alarm by e.g. increased system load is raised. The external security rule server actively investigates by modifying the environment causing an intruder to show suspicious behaviour.

Table 7.3: Comparison of the Different Set-ups

7.6 Summary

This chapter has evaluated the ASCap framework, by discussing each of the design goals identified by our access control model and mechanism comparison. First, the performance overhead of the prototype implementation was discussed, then the scalability. Then qualitative properties, such as flexibility, administration, adaptability, unified interface and that the ASCap proxy does not need to be part of the TCB, were evaluated. The flexibility was evaluated informally by showing how different access control models can be instantiated. In the formal evaluation section a formal proof of the flexibility was given. Then in the final section we showed additional benefits of the ASCap framework, which are outside the scope of the original specification. Using the ability to support partial outsourcing and interface with other IT systems, we developed new intrusion detection strategies. These are different to traditional strategies, which rely on an attack signature database and passively scanning the network traffic. Our strategies allow reaction to anomaly signs by actively alter the system to provoke the attacker to show properties which normally would stay hidden, ultimately reducing the number of false positives.

Chapter 8

Conclusion

Future applications will be employed in large scale, open distributed systems. A key necessity for the development of these applications will be a flexible, scalable, dynamic access control mechanism. This thesis has reviewed different access control models and identified requirements that need to be addressed. It is the conclusion of this work that the combination of the concepts of active capabilities, proxy principle and external security servers allow the design of an access control mechanism which is flexible enough to support a wide range of access control models. Based on these concepts a novel access control mechanism, called the ASCap framework, has been designed and implemented as part of the work presented in this thesis. Its evaluation has been provided through the implementation of examples and a formal model. Finally, the framework has been shown to provide additional benefits outside the area of access control.

This chapter concludes this thesis. Section 8.1 highlights the contributions of this work. Section 8.2 discusses the results of the evaluation. Section 8.3 suggests some future directions that will exploit the benefits of the ASCap framework. Section 8.4 provides a closing remark on this thesis.

8.1 Summary of Contributions

The main contribution is the proposal of a unified access control mechanism, the ASCap framework. Other major contributions are:

Extension and combination of the active capability and hidden software capability approaches to form the active software capabilities: Providing the framework with the property to dynamically adapt the access control model for administrative needs, for example companies being able to change their access control model if their business model changes.

The use of external security servers to support flexibility, scaling and partial outsourcing: Flexibility allowing a company to decide on an access control model per user (group) fully depicting the current marketing strategies.

Using the π-calculus to show that the ASCap framework does not change the instantiated access control model: The proof itself contributes to the formal methods research body, by showing a way to use a behavioural calculus to prove flexibility.

Introduction of partial outsourcing, a new paradigm for access control, that allows parts of the access control decision process to be outsourced to domain experts: Not only joined-access-control-administration becomes possible, but also there are benefits in non-access control areas. For example we show how new *active* intrusion detection strategies can be derived, which unmask hidden intruders.

8.2 Discussion of Evaluation

Our vision of future applications has provided us with the requirements listed in Section 1.1. In the evaluation each requirement was discussed. The performance overhead was compared to that of SSL, which allows us to estimate the cost of session set-up and cryptographical performance costs using the prototype implementation. The result showed a performance overhead comparable to other frameworks and acceptable in relation to the added benefits.

Scaling factors also influence whether a proposed mechanism finds wide acceptance. The ASCap framework provides two scaling enhancing facilities: the possibility of shifting workload to the client application, and providing the external security server interface, which facilitates the use of well-known scaling techniques. Therefore we believe the ASCap framework could also be used to provide access control at operating system level or for Internet-scale distributed applications.

A major goal in the design was to avoid the necessity of making the client part of the framework part of the TCB, according to the open distributed systems assumption presented in Section 2.4. The evaluation used an attack tree-based [146] method, similar to that used in system verification. In order to instantiate highly secure set-ups, it has been shown that the security of the ASCap framework does not rely on a TCB on the client side, but that external security servers can provide the required properties. This result is in line with past results, which employed F-Boxes (AMOEBA, c.f. 3.4.1) or CR provider (OASIS, c.f. 3.4.6).

In the early stages of the research we aimed to extend the constraint of untrusted clients to any trust relationship between the client and the server but it became clear that server authentication on the client side is desirable. Therefore we assumed that the client is able to authenticate the ASCap proxy, e.g. by means of code signing and using a PKI. In a PKI each server's public key needs to be manually set up by the user either by importing it directly from the server CA or via a web of trust. Recent

research in trust-based computing allows establishing trust automatically even if no manual trust relationship is set up. This will be done by observing the environment and building up trust in recognised entities [149].

The ASCap framework provides flexibility which guarantees companies a return on investment, as today's applications can be adapted to tomorrow's security models.

We showed flexibility using two methods: an informal evaluation consisted of different access control set-ups; and a formal evaluation utilised the π-calculus. The π-calculus has been used to describe complex systems, such as the Spanish Fishmarket [129] or NASA's LOGOS[1] multi-agent system [38], but we are aware of no other work using it to prove a system's flexibility. This result is useful in terms of showing the ASCap framework's flexibility, but also may lead to the application of the π-calculus to other domains. Using the two-step process of our proof generalised to enable reasoning about other properties than behaviour.

The evaluation of adaptability and properties of administration was demonstrated by showing how, using external security servers, dynamic policy changes (adaptability) and partial outsourcing (administration) become possible. Dynamic policy changes allow the access control model to be updated without any change to the application. Hence the ASCap framework can be employed in server environments with 24/7 availability.

Different categories of partial outsourcing have been defined during our research (cf. 5.8), depending on the power an external security server may acquire. An external rule server enables a new range of interaction relationships. The administration server is the element of the ASCap framework which controls the level of outsourcing allowing the system to benefit from external expertise holders safely. Frameworks like the simple active capabilities [131] do not retain the same control over the level of outsourcing.

Finally partial outsourcing was evaluated by using the paradigm to design novel

[1]Light Out Ground Operation System

intrusion detection strategies. These strategies allow us to unveil hidden properties of intruders by changing the access control environment in a subtle way. Partial outsourcing leaves room for future research.

8.3 Future Work

The most salient area of future work is the one related to partial outsourcing. One open issue is how to find additional benefits of the paradigm in areas not related to access control. It is conceivable that a model of partial outsourcing enables us to describe relationships during interoperation of access control and non-access control frameworks. Once the interaction patterns are standardised it will be possible to interconnect systems whenever a real world situation requires it.

Future large scale, heterogenous applications may be too complex to be verified formally. Bruce Schneier phrases this in his book: "Digital system security is the same way: We can design idealised operating systems that are provably secure, but we cannot actually build them to work securely in the real world."([148]page 8). During the course of his book he envisions a systematic approach to computer security. The external security servers of the ASCap framework provide such elements, which could be introduced into the system to provide sanity checks. An open question is how their benefit can be proven convincingly.

Also the ASCap framework itself provides room for progression. In this thesis we reviewed access control policy research as a 'customer', whose demands need to be met. Taking advantage of policy languages the policy object generation could be automated to a wider extent. It is conceivable that different policy languages may be employed each time depending on suitability for the application domain. This would allow the

ASCap framework to be integrated into network management tools, such as IBM's Tivoli[2].

The use of JINI provided a convenient way of implementing a prototype, but does not cover the full breath of business applications. Therefore one next step will be to provide an API for different programming languages with the purpose of standardising the ASCap proxy download and deployment. However, the diversity of client and server operating systems suggests that the ASCap framework should not aim to become a middleware replacement, but rather a plug-in for existing solutions. Thus we see the ASCap format and access cycle as the core element best standardised by, for example, an RFC.

Finally, a natural development will be to apply the ASCap framework to as many applications as possible, always aiming at establishing it in emerging application domains. The area of pervasive computing has recently generated much research interest. Context models are being developed which describe the structure of pervasive computing environments. The ASCap framework is able to provide the adaptability and flexibility needed to instantiate context changes as discussed in [35].

8.4 After Thought

As our work was built on different approaches in the area of access control, operating system design and object oriented programming. We see its results not only being influential on access control mechanism, but also on the whole security systems design of tomorrow's open distributed systems.

[2]See http://www.tivoli.com

Appendix A

Definitions of the Π-Calculus

A.1 Formal Foundation

The task is to prove that a certain access control mechanism design does not influence the upper lying access control security model. It can be done in different ways. The use of classical Hoare triples [61] would allow to verify that certain properties of programs hold. The number of properties and type would cause the full proof to be complex and very specific. Another approach combined logical reasoning with calculus for communication systems (CCS) to capture the security properties, while benefit from the ease of expressing the communication between different system entities. The idea of having an object (i.e. policy) in one scope (ASCap proxy) and only after some communications being accessible by other objects (such as server) would, however, be particularly hard to encode in the CCS.

This brought us to the π-calculus, which was developed to reason about the behaviour of concurrent communication and mobile systems. The notion of restriction and scope, turned out to be ideal to express our dynamic system.

A.1.1 The π-Calculus

In the following we will give a definition of reduction rules and axioms used in the formal proof in Section 7.4. We will follow the notation of Sangiorgi and Walker [143] and benefit from further insights of Milners introduction [100]. All theorems were taken out of Sangiorgi and Walker's book [143].

A.1.2 Syntax

Let us write $L(0, \pi, +, =, |, \nu)$ for the language of terms given by

$$P ::= 0 \quad | \quad \pi.P \quad | \quad P + P' \quad | \quad \varphi P P' \quad | \quad P|P' \quad | \quad \nu z P$$

where the prefixes are given by

$$\pi ::= \overline{x}y \quad | \quad x(z) \quad | \quad \tau$$

and the conditions by

$$\varphi ::= [x = y] \quad | \quad \neg\varphi \quad | \quad \varphi \wedge \varphi'.$$

Here $\overline{x}y$ means sending y on the channel x; x(z) refers to receiving a value on channel x, which is then bound to z and τ is the internal transition. Finally to ease notation we abbreviate $(\nu z)\overline{x}z$ by $\overline{x}(z)$.

A.1.3 Labelled Transition Relations

Each system described by a *process expression* can transit into other process expression by the transition relations given below. We will use the late transition rules, which are distinguished by the time when the placeholder z is instantiated. The instantiates

occurres late, when the *communication* is inferred, rather than when the *input* by the receiver is inferred.

Definition A.1.1. Late transition relations *The late transition relations,* $\{\overset{\alpha}{\longmapsto} | \alpha \in \pi\}$, *are defined by the rules in Table A.1.*

THE LATE TRANSITION RULES

$$\text{L-OUT}\ \frac{}{\overline{x}y.P \overset{\overline{x}y}{\longmapsto} P} \qquad \text{L-INP}\ \frac{}{x(z).P \overset{x(z)}{\longmapsto} P}$$

$$\text{L-TAU}\ \frac{}{\tau.P \overset{\tau}{\longmapsto} P} \qquad \text{L-MAT}\ \frac{\pi.P \overset{\alpha}{\longmapsto} P'}{[x=x]\pi.P \overset{\alpha}{\longmapsto} P'}$$

$$\text{L-SUM-L}\ \frac{P \overset{\alpha}{\longmapsto} P'}{P+Q \overset{\alpha}{\longmapsto} P'}$$

$$\text{L-PAR-L}\ \frac{P \overset{\alpha}{\longmapsto} P'}{P|Q \overset{\alpha}{\longmapsto} P'|Q}\ bn(\alpha) \cap fn(Q) = 0$$

$$\text{L-COMM-L}\ \frac{P \overset{\overline{x}y}{\longmapsto} P' \quad Q \overset{x(z)}{\longmapsto} Q'}{P|Q \overset{\tau}{\longmapsto} P'|Q'\{y/z\}}$$

$$\text{L-CLOSE-L}\ \frac{P \overset{\overline{x}(z)}{\longmapsto} P' \quad Q \overset{x(z)}{\longmapsto} Q'}{P|Q \overset{\tau}{\longmapsto} \nu z(P'|Q')}$$

$$\text{L-RES}\ \frac{P \overset{\alpha}{\longmapsto} P'}{\nu z P \overset{\alpha}{\longmapsto} \nu z P'} z \notin n(\alpha) \qquad \text{L-OPEN}\ \frac{P \overset{\overline{x}z}{\longmapsto} P'}{\nu z P \overset{\overline{x}(z)}{\longmapsto} P'}\ z \neq x$$

$$\text{L-REP-ACT}\ \frac{P \overset{\alpha}{\longmapsto} P'}{!P \overset{\alpha}{\longmapsto} P'|!P}$$

$$\text{L-REP-COMM}\ \frac{P \overset{\overline{x}y}{\longmapsto} P' \quad P \overset{x(z)}{\longmapsto} P''}{!P \overset{\tau}{\longmapsto} (P'|P''\{y/z\})|!P}$$

$$\text{L-REP-CLOSE}\ \frac{P \overset{\overline{x}(z)}{\longmapsto} P' \quad P \overset{x(z)}{\longmapsto} P''}{!P \overset{\tau}{\longmapsto} (\nu z(P'|P''))|!P}$$

Table A.1: The Late Transition Rules

We shall define an additional transition rules for φPQ (mismatch operator).

$$MISM1: \quad \frac{P \overset{\alpha}{\longmapsto} P' \quad [|\varphi|]=true}{\varphi PQ \overset{\alpha}{\longmapsto} P'}$$

$$MISM2: \quad \frac{Q \overset{\alpha}{\longmapsto} Q' \quad [|\varphi|]=false}{\varphi PQ \overset{\alpha}{\longmapsto} Q'}$$

A.1.4 Behavioural Equivalence

The basic equivalence is the structural congruence, which is defined as:

Definition A.1.2. Structural Congruence Structural congruence, *written* $\equiv$, *is the process congruence over* P *determined by the following equations:*

1. *Change of bound names (alpha-conversion)*

2. *Reordering of terms in a summation*

3. $P|0 \equiv P, P|Q \equiv Q|P, P|(Q|R) \equiv (P|Q)|R$

4. $\nu a(P|Q) \equiv P|\nu aQ$ *if* $a \notin fn(P)$

5. $\nu a0 \equiv 0$, $\nu abP \equiv \nu baP$

In the literature other forms of behavioural equivalence have been described. We are going to restriction us to the form of weak late congruence, which has a stronger notion than weak late bisimilarity.

Definition A.1.3. Weak late bisimilarity *Weak late bisimilarity is the largest symmetric relation,* $\approx_l$*, such that whenever* $P \approx_l Q$*,*

1. $P \overset{x(z)}{\longmapsto} P'$ *implies there is* Q' *such that* $Q \overset{x(z)}{\longmapsto} Q'$ *and* $P'\{y/z\} \approx_l Q'\{y/z\}$ *for every* y

2. *if* α *is not an input action then* $P \overset{\alpha}{\longmapsto} P'$ *implies* $Q \overset{\alpha}{\longmapsto} \approx_l P'$.

Definition A.1.4. Weak late congruence P *and* Q *are weak late congruent,* $P \approx_l^c Q$*, if* $P\sigma \approx_l Q\sigma$ *for every substitution* σ*.*

A.1.5 Proof System

We also record and name the rules of equational reasoning for this language:

$$
\begin{array}{lllll}
REFL & P = P & & & \\
SYMM & P = Q & implies & Q = P & \\
TRANS & P = Q & and \quad Q = R & implies & P = R \\
SUM & P = Q & implies & P + R = Q + R. &
\end{array}
$$

To reason about bisimilarity of summations we add axioms saying that + is associative, commutative, idempotent, and 0 has an identity:

$$\begin{array}{lrcl} S1 & (P+Q)+R & = & P+(Q+R) \\ S2 & P+Q & = & Q+P \\ S3 & P+0 & = & P \\ S4 & P+P & = & P. \end{array}$$

Now consider late bisimilarity on $L(0,\pi,+,=)$. For the input prefix we need to separate out the sound part,

$$PRE1 \quad P=Q \quad implies \quad \pi.P=\pi.Q \quad if \pi \; is \; not \; of \; the \; form \; x(z),$$

and introduce the rule:

$$PRE2 \quad P\{y/z\}=Q\{y/z\} \quad for \; all \quad y \in fn(P,Q,z) \quad implies \quad x(z).P=x(z).Q.$$

For conditions we have to formulate the simple congruence rule

$$C4 \quad P = Q \quad implies \quad \varphi P = \varphi Q.$$

Then we have an axiom that allows us to replace a condition by an equivalent one

$$C5 \quad \varphi P = \varphi' P \quad if \varphi \Leftrightarrow \varphi'$$

and four axioms for conditional forms:

$$\begin{array}{lrcl} C6 & \neg[x=x]P & = & \neg[x=x]Q \\ C7 & \varphi PP & = & P \\ C8 & \varphi PQ & = & \neg\varphi QP \\ C9 & \varphi(\varphi' P) & = & (\varphi \wedge \varphi')P \end{array}$$

Then we have a kind of distribution axiom involving conditions and summations:

$$C10 \quad \varphi(P+P')(Q+Q') = \varphi PQ + \varphi P'Q'$$

And finally, we have two axioms concerning conditions and prefixing, namely

$$\begin{array}{lrcl} C11 & \varphi(\pi.P) & = & \varphi\pi.\varphi P \\ C12 & [x=y](\pi.P) & = & [x=y](\pi\{y/x\}).P \end{array}$$

We need some axioms for expanding compositions and for manipulating restrictions. Consider expansion first. The axiom will be called **E**:
If $P = \sum_i \varphi_i\pi_i.P_i$ and $P' = \sum_j \varphi'_j\pi'_j.P'_j$, then

$$\begin{aligned} P|P' \;=\; & \sum\nolimits_i \varphi_i\pi_i.(P_i|P') + \sum\nolimits_j \varphi'_j\pi'_j.(P|P'_j) + \\ & \sum\nolimits_{\pi_i \mathrm{opp} \pi'_j} (\varphi_i \wedge \varphi'_j \wedge [x_i = x'_j])\tau.R_{ij} \end{aligned}$$

where $\pi_i \mathrm{opp}^1 \pi'_j$ if

(1) π_i is $\overline{x_i}y$ and π'_j is $x'_j(z)$, when R_{ij} is $P_i|P'_j\{y/z\}$,or

(2) π_i is $\overline{x_i}(z)$ and π'_j is $x'_j(z)$, when R_{ij} is $\nu z(P_i|P'_j)$,

or vice versa. Now we can introduce the rules for restrictions:

$$RES \quad P = Q \quad implies \quad \nu zP = \nu zQ$$

$$\begin{array}{lrcl} RES1 & \nu z\nu wP & = & \nu w\nu zP \\ RES2 & \nu z(P+Q) & = & \nu zP + \nu zQ \\ RES3 & \nu z\pi.P & = & \pi.\nu zP \quad if z \notin n(\pi) \\ RES4 & \nu z\pi.P & = & 0 \quad \text{if } \pi \text{ is } z(w) \text{ or } \overline{z}y \text{ or } \overline{z}(w) \\ RES5 & \nu z[x=y]P & = & [x=y]\nu zP \quad \text{if } x, y \neq z \\ RES6 & \nu z[z=y]P & = & 0 \quad \text{if } y \neq z. \end{array}$$

[1] read opposes

Then to obtain axiomatisations for *weak* late congruence on $L(0, \pi, +, \varphi)$, it suffices to add the following axioms

$$\begin{array}{lrcl} TAU1 & \pi.\tau.P & = & \pi.P \\ TAU2 & \tau.P + P & = & \tau.P \\ TAU3 & \pi.(P + \tau.Q) & = & \pi.(P + \tau.Q) + \pi.Q. \end{array}$$

Finally, an axiom for replication:

$$REP \quad !(P|Q) = (!P)|(!Q)$$

Let LD be the collection of axioms and rules: REFL, SYMM, TRANS, SUM, S1-S4, C4-C12, E, RES, RES1-RES6, TAU1, TAU2, TAU3 and REP.

Theorem A.1.1. *LD is an axiomatisation for weak late congruence of the terms in* $L(0, \pi, +, \varphi, |, \nu)$.

The proofs for the theorems and rule applications can be found in Sangiorgi and Walker [143].

Bibliography

[1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Fourth ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, April 1997.

[2] M. Abadi, E. Wobber, M. Burrows, and B. Lampson. Authentication in the taos operating system. In *Proceedings of the 14th ACM Symposium on Operating System Principles*, pages 256–269, The Grove Park Inn and Country Club, Asheville, NC, December 1993. ACM Press.

[3] J. Abendroth. Applying π-calculus to practice: An example of a unified security mechanism. Technical Report RS-03-39, Basic Research In Computer Science, University of Aarhus, November 2003.

[4] J. Abendroth and C. D. Jensen. Partial outsourcing: A new paradigm for access control. In *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies(SACMAT 03), Como Italy*, pages 134–141, June 2003.

[5] J. Abendroth and C. D. Jensen. A unified security mechanism for networked applications. In *Proceedings of 18th Symposium on Applied Computing (SAC2003)*, pages 351–357. ACM, March 2003.

[6] E. G. Amoroso. *Fundamentals of computer security technology.* Prentice-Hall, Inc., 1994.

[7] J. P. Anderson. Computer Security Technology Planning Study. Technical report, Hanscom AFB, Bedford, MA, October 1972.

[8] B. S. at Counterpane. *Outsourcing Security.* Counterpane webside http://www.counterpane.com/literature.html, 1.12.2002.

[9] R. Au, M. Looi, and P. Ashley. Cross-domain one-shot authorization using smart cards. In *Proceedings of the 7th ACM conference on Computer and communications security*, pages 220–227. ACM Press, 2000.

[10] R. W. Baldwin. Naming and grouping privileges to simplify security management in large database. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland, CA)*, pages 116–132, Los Alamitos, CA, May 1990. IEEE Computer Society Press.

[11] J.-P. Banatre, C. Bryce, and D. L. M'etayer. Compile-time detection of information flow in sequential programs. In *Proceedings of the Third European Symposium on Research in Computer Security*, pages 55–73. Springer-Verlag, november 1994.

[12] D. Bell and L. LaPadula. Secure computer systems: Mathematical foundations and model. Report MTR 2547 vII, MITRE Corporation, Bedford, MA, November 1973.

[13] D. Bell and L. LaPadula. Secure computer systems: Unified exposition and multics interpretation. Report ESD-TR-75-306 MTR 2997 v1, MITRE, March 1976.

[14] D. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations. Technical Report MTR-2547, Vol. 1, MITRE Corp., Bedford, MA, March 1973.

[15] S. Benferhat, R. El Baida, and F. Cuppens. A stratification-based approach for handling conflicts in access control. In *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies(SACMAT 03), Como Italy*, pages 189–195, June 2003.

[16] K. Biba. Integrity considerations for secure computer. Technical Report MTR-3153 Rev1 (ESD-TR-76-372), MITRE Corp.Bedfort MA,1976, April 1976.

[17] M. Bishop. Theft of information in the take grant protection model. In *Computer Security Foundations Workshop*, pages 194–218, Rockport, Massachusetts, U.S.A., June 1988.

[18] M. Bishop. Applying the Take-Grant Protection Model. Technical Report PCS-TR90-151, Dartmouth College, Computer Science, Hanover, NH, January 1990.

[19] J. Biskup and S. Wortmann. Towards a credential-based implementation of compound access control policies. In *Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 31–40. ACM Press, 2004.

[20] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164–174, Washington, DC, USA, May 1996. IEEE Computer Society.

[21] W. E. Boebert. On the inability of an unmodified capability system to enforce the *-property. In *In Proceedings of the 7th DoD/NBS Computer Security Conference*, pages 291–293, September 1984.

[22] D. F. C. Brewer and M. J. Nash. The chinese wall security policy. In *IEEE Symposium on Security and Privacy*, pages 206–214, May 1989.

[23] B. Bryant. Designing an authentication system: A dialogue in four scenes. http://web.mit.edu/kerberos/www/dialogue.html, February 1988.

[24] J. A. Bull, L. Gong, and K. R. Sollins. Towards security in an open systems federation. In *European Symposium on Research in Computer Security (ESORICS)*, pages 3–20, November 1992.

[25] R. C, Z. A, and F. P. Spl: An access control language for security policies with complex constraints. In *in Proceedings of the Network and Distributed System Security Symposium (NDSS'01)*, San Diego, California, February 2001.

[26] R. H. Campbell, M. D. Mickunas, and M. Chandak. Sesame authentication protocol. Report UIUCDCS-R-99-2140, University of Illinois atUrbana-Champaign Computer Science Department, June 1999.

[27] D. Clark and D. Wilson. A comparison of commercial and military computer security policies. In *In Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA.*, pages 184–194. IEEE, May 1987.

[28] W. A. S. Consortium. Web application security consortium: Threat classification: Version1.0, August 2004.

[29] J. Daemen and V. Rijmen. Aes proposal: Rijndael, March 2000.

[30] N. Damianou, A. K. Bandara, M. Sloman, and E. C. Lupu. A survey of policy specification approaches, April 2002.

[31] A. Dearle, di Bona, R. Farrow, F. Henskens, D. Hulse, D. Lindstrom, A. Norris, S. Rosenberg, and Vaughan. Protection in the grasshopper operating system. In *Proceedings of the 6th International Workshop on Persistent Object Systems*, pages 60–78, Tarascon, France, September 1994. Springer Verlag.

[32] S. A. Demurjian, 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 J. Weeldreyer, editor, *Proceedings of the 12th Annual International Phoenix*

Conference on Computers and Communications (Tempe, AR), pages 195–202, Los Alamitos, CA, March 1993. IEEE Computer Society Press.

[33] D. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, May 1976.

[34] D. E. Denning. *Secure Information Flow in Computer Systems.* PhD thesis, Purdue University, West Lafayette, IN, May 1975.

[35] S. Dobson and P. Nixon. More principled design of pervasive computing systems. In *Proceedings of Engineering for Human-Computer Interaction and Design, Specification and Verification of Interactive Systems (EHCI-DSVIS'04)*, LNCS. Springer-Verlag, 2004. To appear.

[36] N. Dulay, N. Damianou, E. Lupu, and M. Sloman. A policy language for the management of distributed agents. In *AOSE*, pages 84–100, May 2001.

[37] N. Dunlop, J. Indulska, and K. Raymond. Dynamic conflict detection in policy-based management systems. In *Sixth International ENTERPRISE DISTRIBUTED OBJECT COMPUTING Conference (EDOC'02) September 17 - 20, 2002*, page 15ff, 2002.

[38] A. C. Esterline and T. Rorie. Using the π-calculus to model multiagent systems. In *Lecture Notes in Computer Science Volume 1871*, pages 164–179, April 2001.

[39] R. S. Fabry. Capability–based adressing. *Communications of the ACM*, 17(7):403–412, July 1974.

[40] M. B. J. Feigenbaum and A. D. Keromytis. Keynote: Trust management for public-key infrastructures. In *in Proceedings of the Cambridge 1998 Security Protocols International Workshop, Cambridge, England,LNCS 1550*, pages 59–63, April 1998.

[41] Ferraiolo and Kuhn. Role based access control. In *Proceedings of 15th National Computer Security Conference*, October 1992.

[42] D. F. Ferraiolo, R. S. Sandhu, S. I. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *Information and System Security*, 4(3):224–274, August 2001.

[43] T. Fine and S. E. Minear. Assuring distributed trusted mach. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 206–218, 1993.

[44] S. Fischer-Hbner and A. Ott. From a formal privacy model to its implementation. In *National Information Systems Security Conference (NISSC 98)*, 1998.

[45] E. Freudenthal, T. Pesin, L. Port, E. Keenan, and V. Karamcheti. dRBAC: distributed role-based access control for dynamic coalition environments. In *22nd International Conference on Distributed Computing Systems (ICDCS '02)*, pages 411–420. IEEE, jul 2002.

[46] S. Ganta. *Expressive Power of Access Control Models Based on Propagation of Rights.* PhD thesis, George Mason University, 1996.

[47] D. Gollmann. *Computer Security.* John Wiley & Son Ltd., 1999.

[48] L. Gong. A secure identity-based capability system. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 56–63, May 1989.

[49] A. Goscinski. *Distributed Operating Systems: Logical Design.* Addison-Wesley Publishing Company, Sydney Australia, 1991.

[50] J. Gray. Notes on data base operating systems. In *Lecture Notes In Computer Science 60, Operating Systems, An Advanced Course*, pages 393–481. Springer-Verlag, February 1978.

[51] D. Hagimont, J. Mossiere, X. R. de Pina, and F. Saunier. Hidden software capabilities. In *International Conference on Distributed Computing Systems*, pages 282–289, May 1996.

[52] D. Hagimont and J. Vandewalle. Jccap: Capability-based access control for java card. In *Smart Card Research and advanced Applications - Proceedings of CARDIS'2000.*, 2000.

[53] F. Hantelmann. Benchmarks; sterntaler; alternative auswertung der spec-cpu95-resultate mit starplot. *iX*, 1(1):128, 1999.

[54] A. Harrington and C. Jensen. Cryptographic access control in a distributed file system. In *Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 158–165. ACM Press, June 2003.

[55] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, August 1976.

[56] R. Hayton. *OASIS: An Open Architecture for Secure Interworking Services.* PhD thesis, Univeristy of Cambridge Computer Laboratory, June 1996. Technical Report No. 399.

[57] R. Hayton and K. Moody. An open architecture for secure interworking services. In *ACM SIGOPS European Workshop 96*, September 1996.

[58] M. S. Heng. Understanding electronic commerce from a historical perspective. *Communications of the Association for Information Systems*, 12 Article 6, July 2003.

[59] A. Heydon, M. W. Maimone, J. D. Tygar, J. M. Wing, and A. M. Zaremski. Miro: Visual specification of security. *In IEEE Transactions on Software Engineering*, 16(10):1185–1197, October 1990.

[60] M. Hitchens and V. Varadharajan. Tower: A language for role based access control. In M. Sloman, J. Lobo, and E. Lupu, editors, *Policies for Distributed Systems and Networks, International Workshop, POLICY 2001 Bristol, UK, January 29-31, 2001, Proceedings (LNCS 1995)*, volume 1995 of *Lecture Notes in Computer Science*. Springer, January 2001.

[61] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, October 1969.

[62] C. Hoch and J. C. Browne. An implementation of capabilities on the pdp-11/45. *In Operating Systems Review, ACM SIGOPS*, 14(3):22–32, July 1980.

[63] D. Holden. An exploration of the nature of management policy. Technical Report ESPRIT 5165 harw T2.1 1_0, AEA Industrial Technology, Harwell Laboratory, Oxfordshire, UK,, February 1991.

[64] M. E. Houdek, F. G. Soltis, and R. L. Hoffman. Ibm system/38 support for capability-based addressing. In *In Proceedings of the 8th annual symposium on Computer Architecture*, pages 351–348, Los Alamitos, CA, USA, May 1981. IEEE Computer Society Press.

[65] J. D. Howard. *An analysis of security incidents on the Internet 1989-1995.* PhD thesis, Carnegie Mellon University, 1998.

[66] J. Hughes, C. Feist, S. Hawkinson, J. Perraut, M. O'Keefe, and D. Corcoran. A universal access, smart-card-based, secure file system, 1999.

[67] M. R. A. Huth and M. D. Ryan. *Logic in Computer Science.* Cambridge University Press, The Edinburgh Building, Cambridge, DB2 2RU, UK, November 2002.

[68] T. Jaeger, R. Sailer, and X. Zhang. Resolving constraint conflicts. In *Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 105–114. ACM Press, 2004.

[69] S. Jajodia, P. Samarati, and V. S. Subrahmanian. A logical language for expressing authorizations. In *In Proc. of the IEEE Symposium on Security and Privacy*, pages 31–42, May 1997.

[70] S. Jajodia, P. Samarati, V. S. Subrahmanian, and E. Bertino. A unified framework for enforcing multiple access control policies. In *SIGMOD International Conference on Management of Data*, pages 474–485. ACM Press, May 1997.

[71] W. A. Jansen. A revised model for role-based access control. Technical Report IR 6192, NIST, July 1998.

[72] C. Jensen and D. Hagimont. Protection reconfiguration for reusable software. In *Second Euromicro Conference on Software Maintenance and Reengeneering*, pages 74–81, Florence, Italy, March 1998.

[73] M. K. Johnson. Lurking with pgp. *Linux J.*, 1996(32es):1, 1996.

[74] W. Johnston, S. Mudumbai, and M. Thompson. Authorization and attribute certificates for widely distributed access control. In *Proceedings of the 7th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 98)*, pages 340–345. IEEE Computer Society, June 1998.

[75] A. K. Jones. *Protection in programmed systems.* PhD thesis, Carnegie-Mellon University, June 1973.

[76] D. Jonscher. Extending access control with duties - realized by active mechanisms. In *in Proceedings of IFIP WG11.3 Workshop on Database Sercurity, VI: Status and Prospects*, Washington, DC, August 1992.

[77] D. Jonscher and K. R. Dittrich. Argos – A configurable access control system for interoperable environments. In T. C. Ting and D. Spooner, editors, *Database Security, IX: Status and Prospects*, pages 43–60. Chapman & Hall, 1996.

[78] R. Y. Kain and C. E. Landwehr. On access checking in capability-based systems. In *IEEE Symposium on Security and Privacy*, pages 95–101, 1986.

[79] P. A. Karger and A. J. Herbert. An augmented capability architecture to support lattice security and traceability of access. In *IEEE Symposium on Security and Privacy*, pages 2–12, 1984.

[80] R. Khare and A. Rifkin. Weaving a web of trust. *World Wide Web J.*, 2(3):77–112, summer 1997.

[81] E. J. Khayat and A. E. Abdallah. A formal model for flat role-based access control. In *Proceedings of ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'03)*, July 2003.

[82] L. Kleinrock. Time-shared systems: a theoretical treatment. *J. ACM*, 14(2):242–261, April 1967.

[83] J. Kohl and C. Neumann. The kerberos network authentication service (v5). RFC 1510, Digital Equipment Corporation/ISI, September 1993.

[84] W. Kuehnhauser. Metapolitiken (german). Technical Report RS-AiS-1999-13,ISBN3-88457-362-4, Informationstechnik GmbH. Sankt Augustin, 1999.

[85] W. E. Kuehnhauser. A paradigm for user-defined security policies. In *1995 IEEE Symposium on Reliable Distribution of Systems*, pages 135–144, September 1995.

[86] W. E. Kuehnhauser. Policy groups. *Computers and Security Journal*, 18(4), 1999.

[87] W. E. Kuehnhauser and M. von Kopp Ostrowski. A framework to support multiple security policies. In *In Proceedings of the 7th Annual Canadian Computer Security Symposium, Canadian System Security Centre, Ottawa, Canada*, May 1995.

[88] M. Kuhlmann, D. Shohat, and G. Schimpf. Role mining - revealing business roles for security administration using data mining technology. In *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies(SACMAT 03), June, Como Italy*, pages 179–185, 2003.

[89] B. Lampson. Protection. In *Proceedings of the 5th Annual Princeton Conference on Information Sciences and Systems*, pages 437–443, Princeton University, March 1971.

[90] B. W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, October 1973.

[91] B. W. Lampson, W. W. Lichtenberger, and M. W. Pirtle. A user machine in a time-sharing system. In *Proceedings of 54, 12 (Dec. 1966)*, pages 1766–1774, 1966.

[92] C. Landwehr. Formal models for computer security. *ACM Computing Surveys*, 13(3):247–278, September 1981.

[93] H. M. Levy. *Capability–Based Computer Systems.* Digital Press, 1984.

[94] G. Lowe. An attack on the needham-schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1995.

[95] D. A. Marriott, M. S. Sloman, and N. Yialelis. Management policy service for distributed systems. Technical Report DoC 95/10, Imperial College, London, October 1995.

[96] M. J. Masullo and S. B. Calo. Policy management: An architecture and approach. In *In IEEE First International Workshop on Systems Management*, Los Angeles, California, April 1993. IEEE.

[97] J. McLean. Security models and information flow. In *IEEE Symposium on Security and Privacy*, pages 180–189, May 1990.

[98] S. Microsystems. J2ee 1.4 specification - final release, 24.11.2003.

[99] J. K. Millen. Security kernel validation in practice. *Commun. ACM*, 19(5):243–250, May 1976.

[100] R. Milner. *Communicating and Mobile Systems: The Pi-Calculus.* Cambridge University Press, The Edinburgh Building, Cambridge, DB2 2RU, UK, 1999.

[101] S. E. Minear. Providing policy control over object operations in aMach-Based system. In *In Proceedings of the Fifth USENIX UNIX Security Symposium*, pages 141–156 (or 141–156), June 1995.

[102] N. H. Minsky and A. D. Lockman. Ensuring integrity by adding obligations to privileges. In *Proceedings of the 8th international conference on Software engineering*, pages 92–102. IEEE Computer Society Press, August 1985.

[103] J. Mirkovic and P. Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *SIGCOMM Comput. Commun. Rev.*, 34(2):39–53, 2004.

[104] J. G. Mitchell, J. Gibbons, G. Hamilton, P. B. Kessler, Y. Y. A. Khalidi, P. Kougiouris, P. Madany, M. N. Nelson, M. L. Powell, and S. R. Radia. An overview of the spring system. In *COMPCON*, pages 122–131, February 1994.

[105] J. D. Moffett and M. S. Sloman. The representation of policies as system objects. *In ACM SIGOIS Bulletin*, 12(2-3):171–184, November 1991.

[106] J. D. Moffett and M. S. Sloman. Policy hierarchies for distributed system management. *IEEE JSAC Special Issue on Network Management*, 11(9), November 1993.

[107] S. J. Mullender and A. S. Tanenbaum. The design of a Capability-Based distributed operating system. *The Computer Journal*, 29(4):289–299, August 1986.

[108] S. J. Mullender, G. van Rossum, A. S. Tanenbaum, R. van Renesse, and H. van Staveren. Amoeba: A distributed operating system for the 1990s. *IEEE Computer*, 23(5):44–53, May 1990.

[109] A. C. Myers. JFlow: Practical mostly-static information flow control. In *Symposium on Principles of Programming Languages (POPL)*, pages 228–241, San Antonio, Texas, January 1999.

[110] G. J. Myers. *Advances in Computer Architecture.* John Wiley & Sons, Inc., 1982.

[111] S. Namasivayam. Profiting from business process outsourcing. *IT Professional*, 6(1):12–18, January 2004.

[112] M. J. Nash and K. R. Poland. Some conundrums concerning separation of duty. In *Proceedings of the 1990 IEEE Conference on Security and Privacy (SSP '90)*, pages 201–209. IEEE, May 1990.

[113] National Institute of Standards and Technology. *Minimum Security Requirements for Mulit-User Operating systems,1992 draft, later FIPS PUB 140.* National Institute for Standards and Technology, Gaithersburg, MD, USA, January 1992.

[114] National Institute of Standards and Technology. *ANSI NCITS 359-2004: Information technology - Role Based Access Control.* National Institute for Standards and Technology, Gaithersburg, MD, USA, February 2004.

[115] J. Nazario. Source code scanners for better code. *Linux Journal, http://www.linuxjournal.com//article.php?sid=5673*, 26 Januar 2002.

[116] R. M. Needham and R. D. Walker. The cambridge cap computer and its protection system. In *In Proceedings of the sixth ACM symposium on Operating systems principles, ACM SIGOPS*, pages 1–10, New York, USA, November 1977. ACM Press.

[117] B. C. Neumann. Proxy-based authorisation and accounting for distributed systems. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, pages 283–291, Pittsburgh, Penn, U.S.A., May 1993.

[118] B. C. Neumann, J. G. Steiner, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *Winter 1988 USENIX Conference*, pages 191–201, Dallas, TX, February 1988.

[119] P. Neumann, R. Boyer, R. Feiertag, K. Levitt, and L. Robinson. A provably secure operating system: The system, its applications, and proofs. Technical Report CSL-116, SRI International, Menlo Park, CA, May 1980. 2^{nd} ed.

[120] M. T. D. Nichols and D. Terry. Delegation through access control programs. In *Proceedings of 12th International Conference on Distributed Systems*, pages 529–536, June 1992.

[121] G. M. Nyanchama and S. L. Osborn. Orthogonal views in object oriented database security. Technical Report 308, The Department of Computer Science The of Computer Science The University of Western Ontario London Ontario N6A 5B7 Canada, March 1994.

[122] M. Nyanchama and S. Osborn. Access rights administration in role-based security systems. In J. Biskup, M. Morgernstern, and C. Landwehr, editors, *Proc. 8th*

IFIP WG 11.3 Working Conference on Database Security (Database Security VIII: Status and Prospects) Bad Salzdetfurth, Germany, Aug. 23–26, volume A-60 of *IFIP Transactions*, Amsterdam, The Netherlands, August 1995. North-Holland (Elsevier).

[123] M. Nyanchama and S. L. Osborn. Role-based security, object oriented databases & separation of duty. *SIGMOD Record*, 22(4):45–51, June 1993.

[124] D. of Trade and I. UK. Net benefit: the electronic commerce agenda for the uk. *DTI UK report*, October 1998.

[125] M. S. Olivier. Towards a configurable security architecture. *Data and Knowledge Engineering*, 38(2):121–145, August 2001.

[126] OMG. Corba specification v2.3, june 2001.

[127] K. Ong and R. Lee. A logic model for maintaining consistency of bureaucratic policies. In *International Conference on System Sciences*, January 1993.

[128] A. Ott. The rule set based access control (rsbac) linux kernel security extension. In *International Linux Kongress 2001 National Information Systems Security Conference (NISSC 98)*, 2001.

[129] J. A. Padget and R. J. Bradford. A pi-calculus model of a spanish fish market - preliminary report. In *AMET1998,LNCS 1571*, pages 166–188, May 1998.

[130] J. S. Park. *As/400 Security in a Client/Server Environment.* John Wiley & Sons, July 1995.

[131] T. Qian and W. Liao. Active capability: An application specific security and protection model. Technical report, University of Illinois at Urbana-Champaign, December 1996.

[132] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-oriented modeling and design.* Prentice-Hall, Inc., 1991.

[133] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE J. Selected Areas in Communications*, 21(1):5–19, January 2003.

[134] J. H. Saltzer. Protection and the control of information sharing in multics. *Commun. ACM*, 17(7):388–402, July 1974.

[135] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, pages 1238–1308, September 1975.

[136] R. Sandhu. A lattice interpretation of the chinese wall policy. In *Proceedings of the 15th NIST-NCSC National Computer Security Conference*, pages 329–339, October 1992.

[137] R. Sandhu. Access control: The neglected frontier. In *First Australian Conference on Information Security and Privacy*, Wollong, Australia, June 1996.

[138] R. Sandhu. Rationale for the rbac96 family of access control models, November 1997.

[139] R. Sandhu. Role activation hierarchies. In *Proceedings of the third ACM workshop on Role-based access control*, pages 33–40. ACM Press, October 1998.

[140] R. Sandhu and J. S. Park. Decentralized user-role assignment for web-based intranets. In *Proceedings of the third ACM workshop on Role-based access control*, pages 1–12. ACM Press, October 1998.

[141] R. S. Sandhu. The typed access matrix model. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 122–136, May 1992.

[142] R. S. Sandhu, E. J. Coyne, and H. L. F. and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[143] D. Sangiorgi and D. Walker. *The Pi-Calculus, A theory of Mobile Processes.* Cambridge University Press, The Edinburgh Building, Cambridge, DB2 2RU, UK, 2001.

[144] S. W. Schmitz. The effects of e-commerce on the structure of intermediation. *Journal of Computer Mediated Communication 5*, September 2002.

[145] B. Schneier. Description of a new variable-length key, 64-bit block cipher (Blowfish). *Lecture Notes in Computer Science*, 809:191–204, August 1994.

[146] B. Schneier. Attack trees. *Dr. Dobb's Journal of Software Tools*, 12(4):21–29, december 1999.

[147] B. Schneier. The case for outsourcing security. *IEEE Security and Privacy*, April 2002.

[148] B. Schneier. *Secret and Lies.* John Wiley & Sons; ISBN: 0471253111, August 2000.

[149] J. Seigneur, S. Farrell, C. Jensen, E. Gray, and Y. Chen. End-to-end trust starts with recognition. *LNCS 2802, Proceedings of the First International Conference on Security in Pervasive Computing*, 2003.

[150] M. Shapiro. Structure and encapsulation in distributed systems: The proxy principle. In *Proceedings of the 6th International Conference on Distributed Computer Systems*, pages 198–204, Cambridge, Massachusetts, U.S.A., May 1986.

[151] M. Shapiro, Smith, and Farber. EROS: a fast capability system. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP'99)*, pages 170–185. Kiawah Island Resort, near Charleston, South Carolina, December 1999.

[152] M. Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, **2**:333, December 1994.

[153] O. Source. ant, http://ant.apache.org/, June 2004.

[154] R. Spencer, S. Smalley, P. Loscocco, M. Hibler, D. Andersen, and J. Lepreau. The Flask security architecture: System support for diverse security policies. In *Proc. of the 8th Usenix Security Symposium*, pages 123–139, August 1999.

[155] A. Sundarajan and A. Seidmann. Sharing logistics information across organizations: Technology, competition and contracting. In *Information Technology and Industrial Competitiveness: How IT Shapes Competition*, pages 107–136. Kluwer Academic Publishers, Boston, MA, 1998.

[156] A. S. Tanenbaum, M. F. Kaashoek, R. van Renesse, and H. E. Bal. The amoeba distributed operating system: a status report. *Computer Communications*, 14:324–335, July 1991.

[157] A. S. Tanenbaum, S. J. Mullender, and R. van Renesse. Using sparse capabilities in a distributed operating system. In *Proceedings of the 6th International Conference on Distributed Computing Systems (ICDCS)*, pages 558–563, Washington, DC, June 1986. IEEE Computer Society.

[158] J.-E. J. Tevis and J. A. Hamilton. Methods for the prevention, detection and removal of software security vulnerabilities. In *Proceedings of the 42nd annual Southeast regional conference*, pages 197–202. ACM Press, April 2004.

[159] M. Thompson, W. Johnston, S. Mudumbai, G. Hoo, K. Jackson, and A. Essiari. Certificate-based access control for widely distributed resources. In *Proceedings of the Eighth USENIX Security Symposium (Security 99)*, pages 215–228, August 1999.

[160] B. Turvey. *Criminal Profiling: An Introduction To Behavioral Evidence Analysis.* Academic Press, 24-28 Oval Road, London NW1 7DX,UK, 1999.

[161] J. D. Tygar and J. M. Wing. Visual specification of security constraints. In *Proceedings of the 1987 Workshop on Visual Languages*, pages 288–301. IEEE Computer Society Press, August 1987.

[162] W. P. . G. . E. Union. Recommendation 3/97, anonymity on the internet, December 1997.

[163] Various. Open source pki book, http://opensourcepkibook.sourceforge.net, 1.12.2002.

[164] Various. Dtos formal security policy model (fspm) (non-z version). Technical report, Secure Computing Corporation, DTOS CDRL A019, 2675 Long Lake Rd, Roseville, MN 55113, June 1997.

[165] J. Vochteloo, S. Russell, and G. Heiser. Capability-based protection in the Mungi operating system. In *Proceedings of the Third International Workshop on Object Orientation in Operating Systems*, pages 108–115, December 1993.

[166] P. Wainewritght. Web services infrastructure: The global utility for real-time business, February 2002.

[167] K. G. Walter, S. I. Schaen, W. F. Ogden, W. C. Rounds, D. G. Shumway, D. D. Schaeffer, K. J. Biba, F. T. Bradshaw, S. R. Ames, and J. M. Gilligan. Structured specification of a security kernel. In *Proceedings of the international conference on Reliable software*, pages 285–293, April 1975.

[168] C. Weissman. Security controls in the ADEPT-50 time sharing system. In *In AFIPS Conference Proceedings, Volume 35*, New Jersey, USA, 1969.

[169] R. Wies. Using a classification of management policies for policy specification and policy transformation. In A. S. Sethi, Y. Raynaud, and F. Fure-Vincent, editors, *Integrated Network Management IV*, volume 4, pages 44–56, Santa Barbara, CA, May 1995. Chapman & Hall.

[170] M. V. Wilkes. *Time Sharing Computer Systems.* Elsevier Science Inc., 1975.

[171] T. Y. C. Woo and S. S. Lam. Authorization in distributed systems : A formal approach. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 33–51, 1992.

[172] T. Ylonen. SSH - secure login connections over the internet. In *Proceedings of the 6th Security Symposium) (USENIX Association: Berkeley, CA)*, pages 37–42, July 1996.