The South African Institute for Computer Scientists and Information Technologists

ANNUAL RESEARCH AND DEVELOPMENT SYMPOSIUM

23-24 NOVEMBER 1998
CAPE TOWN
Van Riebeeck Hotel in Gordons Bay

Hosted by the University of Cape Town in association with the CSSA,
Professors from University for CHE, and
The University of Natal

PROCEEDINGS

EDITED BY
D. PEFKOV AND L. VENTER

SPONSORED BY

ABSAGroup
The South African Institute for Computer Scientists and
Information Technologists

ANNUAL RESEARCH AND DEVELOPMENT
SYMPOSIUM

23-24 NOVEMBER 1998
CAPE TOWN
Van Riebeeck hotel in Gordons Bay

Hosted by the University of Cape Town in association with the CSSA,
Potchefstroom University for CHE and
The University of Natal

GENERAL CHAIR : PROF G. HATTINGH, PU CHE

PROGRAMME CO-CHAIRS:
PROF. L VENTER, PU CHE (Vaal Triangle), PROF. D. PETKOV, UN-PMB

LOCAL ORGANISING CHAIR: PROF. P. LICKER, UCT - IS

PROCEEDINGS

EDITED BY
D. PETKOV AND L. VENTER

SYMPOSIUM THEME:
Development of a quality academic CS/IS infrastructure in South Africa

SPONSORED BY
Copyrights reside with the original authors who may be contacted directly.

Edited by Prof. D. Petkov and Prof. L. Venter
Van Reebek Hotel, Gordons Bay, 23-24 November 1998


Keywords: Computer Science, Information Systems, Software Engineering.

The views expressed in this book are those of the individual authors and not of the South African Institute for Computer Scientists and Information Technologists.

Office of SAICSIT: Prof. J.M.Hatting, Department of Computer Science and Information Systems, Potchefstroom University for CHE, Private Bag X6001, Potchefstroom, 2520, RSA.

Produced by the Library Copy Centre, University of Natal, Pietermarizburg.
FOREWORD

The South African Institute for Computer Scientists and Information Technologists (SAICSIT) promotes the cooperation of academics and industry in the area of research and development in Computer Science, Information Systems and Technology and Software Engineering. The culmination of its activities throughout the year is the annual research symposium. This book is a collection of papers presented at the 1998 such event taking place on the 23rd and 24th of November in Gordons Bay, Cape Town. The Conference is hosted by the Department of Information Systems, University of Cape Town in cooperation with the Department of Computer Science, Potchefstroom University for CHE and and Department of Computer Science and Information Systems of the University of Natal, Pietermaritzburg.

There are a total of 46 papers. The speakers represent practitioners and academics from all the major Universities and Technikons in the country. The number of industry based authors has increased compared to previous years.

We would like to express our gratitude to the referees and the paper contributors for their hard work on the papers included in this volume. The Organising and Programme Committees would like to thank the keynote speaker, Prof M.C Jackson, Dean, University of Lincolnshire and Humberside, United Kingdom, President of the International Federation for Systems Research as well as the Computer Society of South Africa and The University of Cape Town for the cooperation as well as the management and staff of the Potchefstroom University for CHE and the University of Natal for their support and for making this event a success.

Giel Hattingh, Paul Licker, Lucas Venter and Don Petkov
Table of Contents

Lynette Drevin: Activities of IFIP wg 11.8 (computer security education) & IT related ethics education in Southern Africa 1

Reinhardt A. Botha and Jan H.P. ElofT: exA Security Interpretation of the Workflow Reference Model 3

Willem Krige and Rossouw von Solms: Effective information security monitoring using data logs 9


Carl Papenfus and Reinhardt A. Botha: A shell-based approach to information security 15

Walter Smuts: A 6-Dimensional Security Classification for Information 20

Philip Machanick and Pierre Salverda: Implications of emerging DRAM technologies for the RAM page Memory hierarchy 27

Susan Brown: Practical Experience in Running a Virtual Class to Facilitate On-Campus Under Graduate Teaching 41

H.D. Masethe, T.A Dandadzi: Quality Academic Development of CS/IS Infrastructure in South Africa 49

Philip Machanick: The Skills Hierarchy and Curriculum 54

Theda Thomas: Handling diversity in Information Systems and Computer Science Students: A social Constructivist Perspective 63

Udo Averweg and G J Erwin: Critical success factors for implementation of Decision support systems 70

Magda Huisman: A conceptual model for the adoption and use of case technology 78

Paul S. Licker: A Framework for Information Systems and National Development Research 79

K. Niki Kunene and Don Petkov: On problem structuring in an Electronic Brainstorming (EBS) environment 89
Derek Smith: Characteristics of high-performing Information Systems Project Managers and Project Teams

Lucas Venêr: INST AP: Experiences in building a multimedia application

Scott Hazelhurst, Anton Fatti, and Andrew Henwood: Binary Decision Diagram Representations of Firewall and Router Access Lists

Andre Joubert and Annelie Jordaan: Hardware System interfacing with Delphi 3 to achieve quality academic integration between the fields of Computer Systems and Software Engineering

Borislav Roussev: Experience with Java in an Advanced Operating Systems Module

Conrad Mueller: A Static Programming Paradigm

Sipho Langa: Management Aspects of Client/Server Computing

T Nepal and T Andrew: An Integrated Research Programme in AI applied to Telecommunications at ML Sultan Technikon

Yuri Velinov: Electronic lectures for the mathematical subjects in Computer Science

Philip Machanick: Disk delay lines

D Petkov and O Petkova: One way to make better decisions related to IT Outsourcing

Jay van Zyl: Quality Learning, Learning Quality

Matthew O Adigun: A Case for Reuse Technology as a CS/IS Training Infrastructure

Andy Bytheway and Grant Hearn: Academic CS/IS Infrastructure in South Africa: An exploratory stakeholder perspective

Chantel van Niekerk: The Academic Institution and Software Vendor Partnership

Christopher Chalmers: Quality aspects of the development of a rule-based architecture

Rudi Harmse: Managing large programming classes using computer mediated communication and cognitive modelling techniques
Michael Muller: How to gain Quality when developing a Repository Driven User Interface  184

Elsabe Cloete and Lucas Venter: Reducing Fractal Encoding Complexities  193

Jean Bilbrough and Ian Sanders: Partial Edge Visibility in Linear Time  200

Philip Machanick: Design of a scalable Video on Demand architecture  211

Freddie Janssen: Quality considerations of Real Time access to Multidimensional Matrices  218

Machiel Kruger and Giel Hattingh: A Partitioning Scheme for Solving the Exact k-item 0-1 Knapsack Problem  229

Ian Sanders: Non-orthogonal Ray Guarding  230

Fanie Terblanche and Giel Hattingh: Response surface analysis as a technique for the visualization of linear models and data  236

Olga Petkova and Dewald Roode: A pluralist systemic framework for the evaluation of factors affecting software development productivity  243

Peter Warren and Marcel Viljoen: Design patterns for user interfaces  252

Andre de Waal and Giel Hattingh: Refuting conjectures in first order theories  261

Edna Randiki: Error analysis in Selected Medical Devices and Information Systems  262
A SHELL-BASED APPROACH TO INFORMATION SECURITY

Carl Papenfus and Reinhardt A. Botha.
Faculty of Computer Studies
Port Elizabeth Technikon
Private Bag X6011
Port Elizabeth 6000
{9433961, reinhard}@ml.petech.ac.za

Abstract

Modern computer systems are truly awe-inspiring fetes of detailed design. Millions of transistors perform billions of instructions every second, linking thousands of people from every continent with a remarkable degree of reliability and accuracy. Yet all of this seems insignificant when compared with the complexities found in nature. Numerous design ideas, for example airplanes, have been inspired by nature. The knowledge of the feasibility of flight served as incentive to achieve such design.

This paper will explore some of the similarities between nature and information systems. It will, however, become clear that there is at least one major difference. This difference lies in the instincts of wild animals to protect themselves. Computer systems, however, have no natural protection mechanisms.

Many protection mechanisms have been designed by practitioners in the field of information security. However, current security models are largely based on the domestic protection model. These models can be compared to a shepherd looking after a flock of sheep. Any predator wishing to prey on the flock need only eliminate or avoid the shepherd. A traditional computer system can be compared to a sheep, not having any natural resilience to fend off attacks from predators.

In contrast to this scenario most animals that live in the wild have built in protection mechanisms to address its security needs. The tortoise is a very good example of such an animal, carrying around a shell that is impervious to almost all of the tortoise’s enemies. The shell is very portable and also acts as protection against the elements allowing the tortoise to be a very independent and relatively secure animal.

This paper proposes a model for protecting information based on the security functionality of the tortoise shell. The model draws on the availability of object technology and suggests the use of an autonomous object based shell to provide a form of “self-protection” similar to that of the tortoise. The shell will possess a degree of intelligence that will restrict access to its data based upon the person or mechanism wishing to access it, as well as the perceived safety of the environment in which this access will take place.

Introduction

To survive in nature animals have successfully been defending themselves from predators and harm since time began by using highly specialized, highly effective defensive mechanisms which have become an everyday part of their lives. Modern day networks are increasingly being viewed as very vast intelligent organisms interacting with each other (Louwrens, C. P. & von Solms S. H., 1998).
Current computer security models are largely based on the domestic protection model, depending on a shepherd to look after the flock of animals having no natural resilience to attacks from predators. Although this strategy works well, not all animals live together in a flock. In nature many animals are continually moving from one place to another making it difficult to use the domestic strategy, therefore wild animals were given instincts to help facilitate their survival. Their defense systems often function without the organism being aware of it. It happens independently of what the creature is doing or thinking, as a set of pre-programmed rules that executes the moment a threat is identified.

The tortoise is a very good example of such an animal, carrying around a shell that is impervious to almost all of the tortoise’s enemies. The shell is very portable and also acts as protection against the elements allowing the tortoise to be a very independent and relatively secure animal.

Security Services

If it is possible for the tortoise to benefit from this form of protection, it may also be viable for information to use the same strategy to protect itself when exposed to hostile environments. To test the validity of such an argument, let us consider some important security issues that would have to be catered for by shells in order to make them a viable security mechanism in an information system.

Integrity
Integrity involves the prevention of loss or modification to information due to corruption or unauthorized access (Gulachenski, B. D. & Costa, M. J., 1994). A tortoise has the ability to retract vulnerable parts of its anatomy into a single seamless shell to protect its integrity if attacked. This strategy allows for the non-exposure of weak points that may be exploited by the attacker. It becomes a useless motionless object offering no threat or benefit to anyone.

Confidentiality
Confidentiality dictates that information must be secure or private (Smythe, 1995). The tortoise has the ability to protect its confidentiality by being able to carry its shell around with it. Any time it feels threatened in its environment it simply retracts itself into the shell, becoming an inanimate rock, of little interest to predators unable to penetrate the shell.

Availability
Availability ensures that systems work in a timely manner and services are not denied to authorized users (Forcht, 1996). The tortoise’s shell protects it not only from predators but also from the elements. This allows the tortoise to move freely to any location or environment it wishes to go to. It is also important to note that a tortoise will stay hidden in its shell until there is a reason for it to move or extend a limb.

Identification and Authentication
Identification and authentication require confirmation of the identity of the communicating partners, as well as the source and integrity of the information (Smythe, 1995). A tortoise can authenticate other animals in the standard way through the use of its senses, sight, smell, etc. Another advantage of having a shell is that the tortoise can authenticate animals without extending its head outside the shell. This allows it to be absolutely certain that things are safe without exposing itself to possible injury.

Non-repudiation
Non-repudiation is concerned with preventing users from wrongfully denying a particular action. This includes proof-of-origin and proof-of-delivery (Smythe, 1995). No two animals are exactly the same. Different types and species of animal can be distinguished through their size, shape and marking. Furthermore animal senses have been sharpened to allow them to determine subtle differences in smell and sound to further aid in the identification of other animals.

From the above comparison we can deduce that the use of a tortoise shell model to protect data may be practical, but how would one model a real world object such as an animal in a computer environment?
Modeling Secure Real World Objects

One possible way to overcome modern computer security problems is to search in nature for new ways to protect information (Louwrens, C. P. & von Solms S. H., 1998). Numerous design ideas have been inspired by nature. Many research papers have been published exploring biological models for various robotics systems, smart agents, learning, navigation and intelligence. Objects allow programmers to model real-world objects (Leiwo, J., Gamage, C. & Zheng, Y., 1998). An object is a thing. We live in a world of objects, the computer you use, the car you drive and the house in which you live. Every object has its own characteristics, such as color, shape and weight. Programmers model real-world objects by declaring classes that represent objects.

A class is typically a collection of data, for example characteristics such as color and shape and data functions. The combining of data and functions into one object is called encapsulation. The data is used to describe the various attributes of the object. Two objects of the same class can therefore contain different data, for example object A has color equal to red, while object B has color equal to blue. The data functions or methods are used to query or change the various data attributes of the object it belongs to, for example change color red to blue (Shammas, N. C., 1996).

As mentioned above objects enable you to declare classes that encapsulate data and methods. The shell based object model takes this concept one step further by surrounding the entire object in a protective shell. This is done to allow each object to manage its own security services making it more robust and independent.

Basic security services are essential in large cooperative distributed systems. But computer security policies that are too restrictive can make matters worse that they originally were. For example a new systems administrator may force all users to change their passwords to new randomly generated strings of meaningless letters and numbers. Because the new passwords are difficult for the users to remember, most of them will write their passwords down and store them in a place where they can be easily found. Likewise if information is made difficult to access, people may be tempted to create their own private copies (Van der Linden, P., 1997).

The self-protecting shell model therefore wishes to combine objects and computer security to develop an access control mechanism that makes the data readily available and easy to access while still maintaining a high degree of data protection.

The Anatomy of the Shell-Based Object Model

In order to understand how shell-based objects interact with each other one must first understand what makes them different from standard objects. Shell-based objects consist of four main parts, illustrated in figure 1 namely: the shell, the token, the methods and the data. Each of these sections is discussed in detail below.

![Figure 1: The anatomy of a shell based object](image-url)
The Shell
The shell is used to protect the object from being accessed by unauthorized users and applications. The shell protects the object by encrypting the object's data and all but one of its methods. This is done to prevent unauthorized outside parties from attempting to bypass the object's authentication process. The one method left unencrypted is called the Do_Request method. The Do_Request method is the only method that is active while the object is in its shell. Any time an object wishes to communicate with another object it must communicate with the Do_Request method. All requests are sent in the form of a token.

The Token
The token contains all the information needed for one object to communicate with another object. This information consists of a unique identification number, the object request and the object's authentication certificate. This certificate uses public key cryptography to allow authentication of the object making the request. The certificate is encrypted with the requesting object's private key. Upon receiving the token the Do_Request method of the receiving object will validate the certificate by decrypting it with the requesting object's public key. If this process is successful the Do_Request method will determine which methods the requesting object may use by viewing the object's access control list.

The Methods
Methods are used by the receiving object to query and manipulate its data. The methods are not exposed for usage. Access can only be achieved through the Do_Request method. This is done to prevent other objects from bypassing the token authentication process. If the token of the requesting object is successfully authenticated, the Do_Request method of the receiving object must "expose" all the methods that the requesting object may use. The exposed methods are then used to process the requesting objects request (see figure 2).

The Data
The data contained within the object can be of any format, as the object and shell are only used to store and protect the data. The data contained within the object will never be directly manipulated by the user, but only by the methods directly associated with the object. This further improves security by preventing users from performing unauthorized actions on the data.

Processing a request using Shell-Based Objects
For the purpose of explanation a request for service by object O_i from object O_j will be considered.

O_i generates a token, T={I_i,R,A} where
I_i is the identification for object O_i,
R=(M,p) is the request for method M, using parameters p, and
A={C_i}K_i is some credentials for O_i encrypted using the private key of O_i.
O_i calls the *Do_Request* method passing token T as parameter. On receiving the token O_i determines the "exposable" methods, \( EM_i = \{ M_j \} \) by checking through the access control list for O_i, \( ACL_i \), to find all methods \( M_j \) where \( (I_i, M_j) \in ACL_i \), i.e. 

\[
EM_i = \{ M_j \mid (I_i, M_j) \in ACL_i \}
\]

If the cardinality of \( EM_i \) is greater than zero \( O_i \) knows that \( O_i \) potentially can access certain methods. \( O_i \) can then authenticate the requestor by decrypting A with \( K^{pub}_i \), the public key of \( O_i \). Successful decryption would imply that it was \( O_i \) who sent request token T.

\( O_j \) can now check whether \( M_x \in EM_i \). If so, then \( O_i \) can activate the method \( M_x \) with parameters \( p_i \), otherwise the request is invalid and can be ignored by object \( O_i \). Data extracted by \( M_x \) can now be passed to \( O_i \) by \( O_j \) encrypted with \( K^{pub}_i \). \( O_i \) can thus extract the data using its private key \( K^{priv}_i \).

**Conclusion**

This paper introduced the concept of shell-based security in an object environment. Further research regarding the exact protocols used in such an environment will be needed. It is also foreseen that careful consideration should be taken regarding the incorporation of this shell-based model into distributed object frameworks, such as CORBA.

It is hoped that this article will stimulate thought regarding the autonomy of objects in security and that it will stimulate further research in this regard.

**Bibliography**


